Norwegian University of Life Sciences
The PhD programme in Science and Technology
Faculty of Science and Technology

Philosophiae Doctor (PhD)
Thesis 2022:55

Press ‘Run’ to Improve Mathematical Expertice (PRIME)

Trykk ‘kjør’ for å øke matematisk kompetanse

Morten Munthe

Press ‘Run’ to Improve Mathematical Expertise (PRIME)

Trykk ‘kjør’ for å øke matematisk kompetanse

Philosophiae Doctor (PhD) Thesis

Morten Munthe

Norwegian University of Life Sciences
The PhD programme in Science and Technology
at the
Faculty of Science and Technology

Ås 2022

Thesis number 2022:55
ISSN 1894-6402
ISBN 978-82-575-2004-5

Acknowledgements

There is a myriad of people that deserve thanks for their support and contribution in different ways throughout the last few years. I would like to thank my supervisor Margrethe Naalsund, for patience and understanding when tirelessly trying to guide a student that ventured into the new and often quite perplexing area of mathematical didactics. The combination of guidance and having an open mind to the field of programming, allowed for many valuable discussions regarding the combination programming and mathematics which has been of immense help throughout my work. I would also like to thank my co-supervisor Ulf Geir Indahl, who through discussions of how to implement programming into mathematics and through classroom visits, parted with valuable experience and advise throughout the work.

I would like to thank the educational agency in Oslo, and particularly Kjersti Bjønness and Bjarte Rørmark who supported me in both the application process and with most of the bureaucracy needed to complete my degree. Additionally, my mentor in the educational agency, Knut Skrindo, whose valuable insights into the implementation of programming into classroom helped me both design and revise my work.

Special thanks go to all my colleagues at ILU. Hans Erik Lefdal, who not-so-subtle pushed me towards applying for a PhD and has always had faith in my project. Ellen, with whom many mathematical discussions were had. All the other past and present PhD students at NMBU who have been of immense help in everything from relieving stress to providing guidance. And Hanne, who basically administrated my work life towards the end of my work. The rest of ILU whose support and enthusiasm for my work has driven me forward to where I am today.

Another special thanks go to previous and current colleagues at Oslo Handelsgymnasium. Trond Lien, who helped me initiate and supported me when I applied for the doctorate work. Tone Oksum Eriksen, who supported me from day one. Hermod Haug, for many mathematical puzzle-filled ferry commutes. Vigdís Óskarsdóttir, for continuous support and co-arranging a school excursion to Boston. And everyone else at the science section who supported me throughout my work. Additionally, I would like to thank Lance Olav Eastgate, who through countless didactical discussions presented alternative perspectives on how to teach and implement programming.

Thanks also to my colleges at UiO. Knut Mørken who was always positive and available for rewarding discussions. Cathrine Tellefsen, who believed and gave me confidence in the work together with constructive criticism. And the rest of CCSE who was the pioneers in striving for programming to be included in schools.

My friends of many years, Per Øyvind, whose both educational discussions and stress relieving gaming allowed for many good memories. Mads, Bernt and Jostein who throughout the years provided many hours of enjoyment through various adventures.

Thanks also goes to my family who have supported me throughout. And the person who has suffered the most over these years, Gunhild, who had to live through all the ups and downs of such a work from a supportive perspective. I love you and promise to take you on a vacation … soon.

Table of Contents

List of papers

Article 1
Morten Munthe and Margrethe Naalsund
Designing mathematical programming problems
Submitted to Digital Experiences in Mathematics Education – under 2nd review

Article 2
Morten Munthe
Facilitating exploratory talk through mathematical programming problems
Submitted to NOMAD

Article 3
Morten Munthe
Programming in the mathematics classroom – The types of adversities students encounter
Accepted, to be printed autumn of 2022 in Acta Didactica Norden

Abstract

Press Run to Increase Mathematical Expertise (PRIME) investigates the implementation of programming in the mathematics classroom in upper secondary schools in Norway through the lens of designing mathematical programming tasks. Several countries have recently changed their curriculum to include programming as a part of mathematics. With this change comes the challenge of how to combine programming, as a tool, with mathematics to facilitate mathematical learning. A three-year iteration of task design is conducted in which programming is utilised in mathematics and subsequently implemented in classrooms. The implementation in classrooms is investigated through data collection of audio recordings of the students working on the designed task together with a video recording of their computer screen. The transcript of the data is analysed with a focus on exploratory talk and adversities. Exploratory talk is utilised to investigate how to design tasks facilitating exploratory talk as it is closely connected to learning. Adversities, which can advance and hinder learning, are investigated to inform changes to the task design to promote and limit different types of adversities. Through the investigation, PRIME presents a set of recommendations that could guide the task design of mathematical programming tasks which facilitate learning. (1) Design tasks in which programming is a tool for learning mathematics. (2) Task design should facilitate exploratory talk as this is closely connected to learning. (3) It is advantageous for students to engage in programming after they have learnt a mathematical theme, rather than to use programming to learn a new mathematical theme. (4) Tasks should mitigate non-mathematical adversities, that is, adversities relating to programming specifically. (5) Tasks should have a low floor and a high ceiling so as to facilitate the learning of all students. All of these recommendations are discussed in detail.

Norsk sammendrag

Trykk kjør for å øke matematisk kompetanse (PRIME) undersøker implementeringen av programmering i matematikklasserommet i videregående opplæring i Norge gjennom utforming av matematiske programmeringsoppgaver. Flere land har nylig endret læreplanen til å inkludere programmering som en del av matematikk. Med denne endringen kommer utfordringen av hvordan kombinere programmering som et verktøy i matematikk for å legge til rette for læring i matematikk. En treårig iterasjonssyklus der utforming av matematikkoppgaver med programmering og etterfølgende utprøving i klasserom ble gjennomført. Datainnsamlingen av utprøvingen i klasserom ble gjort gjennom lydopptak av elevene mens de arbeidet med oppgavene sammen med videoopptak av skjermen deres. Transkripsjon av datainnsamlingen ble kodet og analysert med søkelys på utforskende samtaler og hindringer. Utforskende samtaler er brukt da dette har en nær kobling til læring og utforming av oppgavene. Hindringer, som både kan legge til rette for og begrense læring, er undersøkt for å bidra til revideringen av oppgavene med mål om å begrense og promotere ulike former for hindringer. PRIME presenterer til slutt et sett med anbefalinger som kan hjelpe i prosessen med å lage matematiske programmeringsoppgaver som legger til rette for læring. (1) Designoppgaver der programmering er et verktøy for å lære matematikk. (2) Oppgavedesignet burde legge til rette for utforskende samtaler siden dette er nært knyttet til læring. (3) Det er fordelaktig å anvende programmering etter at elevene har lært en matematisk metode, mer enn å bruke programmering til å lære nye matematiske metoder. (4) Oppgavene burde minimere antallet ikke-matematiske hindringer, som ofte relateres til programmering spesifikt. (5) Oppgavene burde ha en lav terskel og et høyt tak for å legge til rette for læring for alle elever. Alle disse anbefalingene er diskutert.

1 Preface

I studied physics at University of Oslo (UiO) from 2002 to 2007, attaining a master's degree; during my studies, several of the courses implemented programming as a natural part of the class. I found programming to be an enjoyable way to visualise and calculate physical phenomena, but its usefulness ended there. I did not find that programming helped me understand physics, but that it was mostly an entertaining diversion from calculating by hand. Since leaving university in 2007, I have primarily taught mathematics, but also natural sciences and physics, at an upper secondary school called Oslo Handelsgymnasium (OHG) in Norway. There was no need for programming, and therefore I did not use it actively for several years. The focus was on teaching the students mathematics and trying my best to relate mathematics to real-world examples. Anyone who has taught mathematics knows that although this can sometimes be easy, it can more often be very challenging.

After teaching for about eight years, I attended a course at Norwegian University of Life Sciences (NMBU) to formalise my teaching qualifications. After finishing my course, I was asked to lecture the following year on the subject of mathematic didactics, which was an enjoyable and rewarding challenge. After two years, a discussion started about whether I would be interested in pursuing a doctorate degree, which, at first, I was reluctant to do.

At that time, I was still teaching at OHG and had recently started teaching a new subject called 'technology and science education' (Teknologi og forskningslære). I enjoyed teaching this course, but it was also very challenging. Since the learning goals for the course were somewhat lenient, I decided that I would incorporate programming into the course. This fitted nicely with its STEM focus (science, technology, engineering, and mathematics), and, from my observations, the students thought learning programming and incorporating it into the science subjects was entertaining and rewarding.

The combination of a possible doctoral degree and the lack of research on programming implemented in mathematics coalesced in my mind into the idea of investigating this area in more detail. I started to investigate how programming could assist students in learning different types of mathematical concepts and

pathways and found an ever-increasing range of possibilities, some of which worked and some of which required a redesign, as explained in greater detail in this thesis.

The work presented in this thesis is a guide through the rather unknown landscape of implementing programming in the mathematics classroom in upper secondary school. It is called Press Run to Increase Mathematical Expertise, or PRIME for short.

"My life is a chip in your pile, ante up"- Setzer Gabbiani (FFVI)

2 Introduction

Press Run to Increase Mathematical Expertise (PRIME), the title of this thesis, is used when referring to this work. The wording of the title is derived from the desire to use programming ('press run') to facilitate students' learning of mathematics ('increase mathematical expertise').

The world is developing very fast, and while it is challenging for teachers to keep up with rapid technological change, they persevere in doing so. To put this change into context, the first handheld calculator was introduced into the world by Canon, Inc. in collaboration with Texas Instruments in 1970 (Demana & Waits, 2000). In 1986, the first graphical calculator was produced by Casio of Japan, and in 1996, Texas Instruments released their first calculator with a computer algebra system (CAS), called TI-92. Increases in processing power and utility have continuously improved calculators and expanded the possibilities of what they can do. Today, calculators and smartphones can perform a variety of different calculations spanning the mathematical curriculum from primary to upper secondary school, such as graphing functions and solving integrals, often displaying the entire solving process. The same development is happening in all aspects of our lives, both professionally and personally. As a result, there is a need for schools to increase the amount and width of information taught to students.

Global technological change drove a political decision in Norway and several other countries (Bocconi et al., 2018) to include programming as an integrated subject in mathematics throughout the school system, from Grade 1 to Grade 13. This integration presents an enormous challenge not only for teachers, but also for the authors of student textbooks and universities' and colleges' teacher training educators. All these actors need to evaluate and consider how to implement programming in mathematics, and, as such, there is a need for research into this area.

With several countries integrating programming with mathematics, one might expect multiple studies to have explored this combination at all levels of the educational system. As of writing (spring 2022), however, little research exists on the implementation of programming into the mathematics classroom in upper secondary school. Such research as exists has mostly been conducted in primary

schools, with a few studies investigating lower secondary and university level. A review article dated 2017 found that, of 139 published studies of technology interventions in mathematics education, none had explored programming in upper secondary schools (Bray & Tangney, 2017). One article investigated programming in upper secondary schools (Psycharis & Kallia, 2017), but its analysis was limited as participating students only undertook a pre/post-test.

More recently, articles have surfaced regarding teachers' attitude towards programming, their expectations of programming in the mathematics classroom, and the challenge of educating them to implement programming (Mozelius et al., 2019; Vinnervik, 2021). Sweden implemented programming in its mathematics curriculum two years before Norway and has therefore recently started to produce relevant research. The combination of mathematics and programming is very specific in both the Norwegian and Swedish systems (Skolverket, 2017; Udir, 2019), and several investigations have interviewed teachers. Findings indicate that teachers are positive and recognise that programming is relevant in relation to the digitalisation of society (Ahmed et al., 2020). One challenge is that neither the Swedish nor Norwegian curriculum specifies exactly where and how programming is to be used, leaving much up to the individual teacher (Ahmed et al., 2020; Misfeldt et al., 2020) . Among the challenges presented in the research is that teachers need to learn programming, how to teach it, and how to implement it in their mathematics classroom, the third of which is the focus of PRIME. As tasks are the main 'thing to do' in the mathematics classroom (Watson et al., 2015), PRIME investigates how tasks combining mathematics with programming can be designed.

2.1 Background to programming and mathematics in schools

The logical build of mathematics has a long history, and Euclid's Elements has, for more than two millennia, been the paradigm of rigorous argumentation (Avigad et al., 2009). Euclid's Elements is a collection of 13 books consisting of definitions, postulates, and mathematical proofs. Most of the theorems appearing within it were not discovered by Euclid, but are, rather, a collection of work from previous mathematicians such as Pythagoras, Hippocrates, and Eudoxus. The Elements are structured in a systematic way whereby the theorems appear in a logical sequence, all deriving from five deceptively simple axioms:

- Things which are equal to the same thing are also equal to each other.
- If equals are added to equals, the wholes are equal.

- If equals be subtracted from equals, the remainders are equal.
- Things which coincide with one another are equal to one another.
- The whole is greater than the part.

The build of Euclid's Elements became the foundation of mathematics in the following centuries. Modern mathematics requires more rigour but is essentially built upon the same ideal of constructing new theorems from existing, proven ones. The logical, sequential build of a mathematical solution is very similar to the logical, sequential structure of a program. As with a mathematical proof, a good mathematical program should be logically built, both structurally and mathematically. The overlap between programming and mathematics is an important starting point for the implementation of programming into schools. A good example of this is the build of a mathematical model which uses both mathematical notation and a programming language (Berry, 2013). Similarities include the formal language, with the precise definition of both syntax and semantics, the idea of a model, the build of the structure, and the process of breaking down a problem into smaller pieces, solving each piece, putting the pieces together to form a solution, and testing and evaluating the solution.

Feurzeig (1969) proposed that programming could contribute to mathematics in several ways, from problem-solving strategies and thinking about algorithms to facilitating experimentation and discussions, and was interested in using programming as a foundation for an integrated course in mathematics . Implementing programming in schools is by no means a new idea. In 1980, Seymour Papert published a book called Mindstorms: Children, computers, and powerful ideas, laying out a simple but far-reaching idea of a mathematical microworld, 'Mathland', where children would program a digital or mechanical turtle moving around. Simple commands such as 'forward 100' and 'right 30' enabled children to create geometric structures and patterns.

The programming language LOGO was specifically designed to accommodate children and their exploration. LOGO (from the Greek logos) was developed by Wally Feurzeig, Seymour Papert, and Cynthia Solomon in 1967. The creation was linked to the academic fields of artificial intelligence, mathematical logic, and developmental psychology. LOGO has not changed much since its conception, although it now exists in many different forms, each with its own flavour. Papert (1980) argues for the implementation of computers and programming in schools and how it can facilitate learning mathematics.

Following the publication of Mindstorm, there was great enthusiasm that LOGO and similar programs would reform mathematical teaching in primary school, but this did not occur (see Misfeldt and Ejsing-Duun (2015). LOGO is a language created primarily for children, and in this thesis the focus is on more advanced text-based programming. Today, an increasing number of countries is once again implementing programming as part of their national curriculum (Skolverket, 2017; Udir, 2019). Arguments for implementing programming include the need to service the existing and future work market, given that the combination of mathematical knowledge and knowledge of programming is viewed as a valuable skill (Gravemeijer et al., 2017) – not necessarily the ability to create complex programs, but the knowledge of how a program works and its affordances and constraints. As the educational system aims to equip students with knowledge for the future, I argue that implementing programming in schools is a natural, but certainly not a simple, step forward.

Previously, the first time many students encountered programming was at university level, where they often took a separate course early in their career called 'programming' or something similar. The course was offered because science students, in particular, would use programming in their studies later on when modelling, running simulations, or performing complicated or multiple calculations for which it was generally necessary to use a computer to derive an answer. With technological advance and the use of student laptops at school, it is natural that technological skills such as programming will trickle down the school system. The science that the students learn during their final two years of school contains copious examples ready made for programming. Examples in physics include simulations of fall, with air resistance and the gravitational forces in space, while examples in biology include big data, with the human genome and protein sequences. In mathematics, there are areas that can facilitate mathematical understanding when introducing programming.

A simple example is probability, where programming allows for Monte Carlo simulations of simple to advanced probability events. A simple event is learning probability and bringing dice into the classroom to visualise that the probability of throwing a six is one sixth. Initially, this can be visualised by giving the students ten dice each and making them throw ten times and record the number of times they receive a six. Thereafter, a table can be created, with the whole class contributing their findings – around 1,000 throws of the dice – and a relative probability can be calculated. With a computer, however, a very short code sequence can yield around one million dice throws per second per student computer. The value of this exercise

is limited since it might be thought that everybody knows that the probability, in this case, is one sixth; however, if the complexity of the question is increased, the value of a computer becomes evident. For example, the question 'If you throw five dice, what is the probability of getting a sum of seventeen?' is very difficult and time-consuming to answer through manual calculation, but relatively simple to simulate using programming. Similarly, programming can be applied to algebra, functions, the derivative and the integral, differential equations, and numerical methods. Numerical methods, in particular, are now possible to implement in secondary school, and PRIME does this through the design of tasks.

The building of a task combining mathematics and programming is challenging as the design depends on the mathematical curriculum of a particular grade and course. In advanced science mathematics, it is easy to think of mathematical themes where programming is beneficial, such as numerical calculations of zero points, the derivatives, or the integral. In lower-level courses, this becomes difficult – not because it cannot be done, but because there are other tools that are much better suited to performing the calculation. With other digital programs such as GeoGebra and Excel available, there is little to be gained by using programming to build a program that plots a linear or polynomial graph or to set up a personal budget. GeoGebra is excellent at visualising and handling graphs (Diković, 2009), and Excel is excellent at handling personal economy (Barreto, 2015). Designing such a task would be to force programming where it is of no benefit, leaving both teachers and students frustrated.

Why, then, use programming instead of other digital tools, such as a calculator and GeoGebra? Several digital tools, perhaps most notably computer algebra systems (CAS), hide certain algorithms and methods from the user to simplify the interface. In CAS, there is often a button with 'x=', which solves the equation but does not display the means by which it does so. This is often referred to as a 'black box' (Buchberger, 1990; Rabardel, 2002). Programming can make this 'black box' more transparent. The transparency comes from the need to decompose a problem and then build a program to solve each individual sub-problem before combining all such sub-problems to complete the solution. The design of mathematical tasks utilising programming facilitates the students' understanding of each part of the mathematical solution. Through the programming, the students understand the principles underlying the mathematical method. From the work presented in Article 2, the students gained a deeper understanding of the numerical bisectional method after building the program, since they built and explored each of the underlying structures that make up the method. Students inputting a function in a CAS setting

and pressing 'solve', in contrast, have little to no understanding of what is happening behind the scenes.

Imagine another example where a student has created a program that solves second-degree equations and is now wondering how to expand the program to solve a second-degree logarithm equation.

$$(\ln x)^2 - \ln x - 6 = 0$$

In a graphing tool, the solution could be to plot the function and note the zero points. In a CAS setting, the solution could be to type in the equation and press 'solve'. In programming, however, an element of exploration and creativity is needed to solve the problem. Most digital tools conform to a given procedure or sequence of button presses. There might be creativity in selecting the method for the solution, but there is little creativity in the execution. In programming, there is a combination of the two. Since a program is logical by nature, the user must build a way for the computer to yield the result they want and be creative so the program does not give false or erroneous solutions. In the example presented, the student will need to understand that the program created cannot yield a negative solution for x and compensate for this in the code. Further, the student needs to build in a two-stage process, where the quadratic equation is used to solve $\ln x$, then expand the program to solve x. This development of a program enables mathematical programming tasks to be more 'translucent' than in many other digital tools. The question remains about the design of the mathematical programming task, which leads to the research question for PRIME:

> *What recommendations could guide the design of tasks in which programming is combined with mathematics to facilitate mathematical learning in upper secondary school?*

Recommendations are a step towards design principles; however, as the research is limited to students from two mathematics classrooms, the use of principles would carry neither reliability nor validity. The research question leaves many aspects for consideration. To create a foundation upon which to build answers, it is therefore divided into a set of sub-questions. The first is how could programming be utilised when implemented in a mathematical task? The aim of this sub-question is to investigate different ways to implement programming and examine how the different utilisations affect student learning. The latter part gives the next question: how do students interact when working on mathematical tasks using programming as a tool? Interactions between students are a significant

indicator for learning, and in PRIME, video is recorded of students working on the tasks to investigate such interaction.

Many factors affect such interactions, leading to the following question: how can the design facilitate exploration and discussion? With the implementation of tasks, PRIME investigates how the task can be designed to facilitate student interaction. There will inevitably be elements preventing or limiting the learning, which leads to the final two questions: what adversities do students encounter when working on mathematical tasks using programming as a tool? and how can such adversities be mitigated? Adversities directly influence the interaction and the next iteration of the task design.

Three articles are presented as part of this thesis. Article 1 researches the design of tasks in which programming is used to facilitate mathematical exploration, discussion, and learning, and Articles 2 and 3 investigate the implementation of such tasks in a mathematics classroom. In more detail, Article 2 focuses on how the tasks facilitate exploratory talk, while Article 3 explores different types of adversities experienced by students as they work on the tasks. By developing a set of recommendations, PRIME aims to contribute to the overriding need for research into implementing programming in the mathematics classroom and how it can be designed to contribute to mathematical learning.

2.2 Structure of PRIME

To build a foundation to investigate the research question, PRIME presents the following structure:

The theory chapter presents theory on mathematical learning and the learning of programming, of which exploratory talk (Mercer, 2005) and adidactical situations (Brousseau, 1997) are central aspects. With student interaction and task design, theory investigating the adversities encountered by students is presented. Finally, as task design is part of the research question, previous research in this field is discussed.

The method chapter presents the research design, in which design-based research (DBR) is used. Thereafter, a large section is devoted to the design of the mathematical programming problems (MPPs) created in PRIME. The implementation is then addressed, covering context, participants, and data collection, after which the collected data are analysed. Finally, different ethical considerations influencing PRIME are presented.

The summary chapter presents the three articles written during the PRIME project, building the foundation for the subsequent discussion.

The discussion chapter explores the research question with reference to both the work described in the articles and PRIME. The presentation takes the form of a set of recommendations, followed by the arguments on which they are based.

The conclusion sums up the discussion and gives a broader view of both programming in schools and possible future work.

3 Theory

Figure 1 shows how the theory is built. As mathematical learning is part of the research question, the view on, and elements of, mathematical learning in PRIME are presented. Programming is used as a tool to promote mathematical learning, and so an overview of the knowledge gained from computational learning is given. Both mathematical learning and learning programming are then merged with theory regarding adversity, which is an oft-discussed theme in both areas of research. Finally, theory of task design is presented together with research linking the previous three sections to such theory.

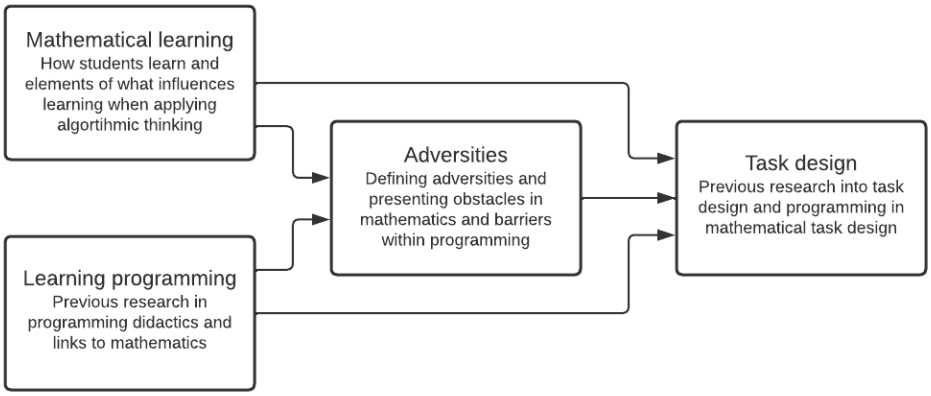


Figure 1: Overview of how the theory is connected to the task design.

3.1 Mathematical learning

Learning in mathematics can be seen as 'the construction of a web of connection – between classes of problems, mathematical objects and relationships, real entities and personal situation-specific experiences' (Noss & Hoyles, 1996, p. 105). This observation is aligned with many other studies using different terms, such as relational understanding (Skemp, 1976), conceptual understanding (Hiebert, 2013), and proception (Gray & Tall, 1994). What these notions all have in common is that they recognise that, ideally, a student forms a network of mathematical concepts so they can draw upon different ones to solve a problem. In

this work, the students gain mathematical understanding through mathematical learning. Mathematical learning is a process, and mathematical understanding is the mental connections between facts, procedures, and ideas (Hiebert & Grouws, 2007). These relationships are reformed when new information is difficult to assimilate or when previous relationships are inadequate to explain a new problem (Piaget, 1964; Skemp, 1976).

When students work on a mathematical task, they will meet resistance and hopefully 'struggle', which is defined by Hiebert and Grouws (2007) as 'expend(ing) effort to make sense of mathematics, to figure something out that is not immediately apparent' (Hiebert & Grouws, 2007, p. 387). The learning takes place when the students are given appropriate tasks or problems. Tasks near the boundaries of a student's zone of proximal development (ZPD) are within reach but offer enough challenge to discover something new (Hiebert & Grouws, 2007; Vygotsky, 1980). This new element can be an increased network or a new concept. The struggle created by the task can be interpreted as productive through Vygotsky's ZPD (Hiebert & Grouws, 2007).

One of the ways to facilitate 'the construction of a web of connection' (Noss & Hoyles, 1996, p. 105) is mathematical interaction with peers (Francisco & Maher, 2005). Research on social interplay, from both constructivist and sociocultural perspectives (Boaler, 1999; Lampert et al., 1996; Lehrer & Schauble, 2005; Resnick et al., 1992; Yackel & Cobb, 1996), and learning though the orchestration of discussion (Michaels et al., 2008) indicates that both facilitate a deep understanding of mathematical concepts of varying difficulty. Explaining reasoning, justifying arguments, and recognising themselves as contributors are part of the exploratory talk between students (Choi & Walters, 2018; Mercer, 2005). Exploratory talk is when 'partners engage critically but constructively with each other's ideas. Statements and suggestions are offered for joint consideration. These may be challenged and counter-challenged, but challenges are justified, and alternative hypotheses are offered. Partners all actively participate, and opinions are sought and considered before decisions are jointly made' (Mercer, 2005, p. 9). The exploratory talk, in turn, increases the students' conceptual understanding, academic vocabulary, and ability to reason in mathematical problem solving (Lampert et al., 1996; Lehrer & Schauble, 2005; Michaels et al., 2008; Resnick et al., 1992; Yackel & Cobb, 1996). The role of mathematics talk and discussion in the effective teaching of mathematics is cemented in research (e.g. Cobb et al., 1997; Kazemi & Stipek, 2009; Nathan & Knuth, 2003; Resnick et al., 2017; Sfard, 2000).

According to such research, one foundation on which PRIME should be based is that exploratory talk promotes mathematical learning. The mathematical tasks using programming facilitate exploratory talk through their design. Other types of talks also occur between students in the classroom, such as talk not relating to mathematics (Ryve, 2011) and disputational and cumulative talk (Mercer, 2005). Talk not relating to mathematics is important for the social setting in the classroom and, by extension, the learning environment. Disputational talk is characterised by disagreement and individual decision making with few attempts to present constructive criticism and suggestions. Cumulative talk occurs when the group constructs common knowledge without criticism, often characterised by repetition and confirmation. While all these types of talk were noted in the data gathered for the present study to varying degree s, the focus for PRIME was facilitating exploratory talk, as this promotes learning.

Mathematical learning and understanding need to be interconnected with programming in the new curriculum. To facilitate exploratory talk through task design, PRIME applies adidactical situations from the theory of didactical situations (TDS) (Brousseau, 1997). Adidactical situations occur when students, working on tasks in small groups, take responsibility for their learning.

According to Brousseau (1997), valuable mathematical learning will more likely take place when the students are committed to a problem situation. In TDS, knowledge is a property of a system consisting of a subject and a *milieu*, or environment. The problem, programming language, and students of a particular group create the milieu with which the latter interact. The interaction can be conceptualised as the conversation within and feedback from the milieu (Brousseau, 1997).

The *didactical contract* is a core concept in TDS and exists between the educator and the students of the milieu. The contract implies a set of expectations regarding the transference of mathematical knowledge and the responsibilities of both the teaching and the learning process. The students expect that, through solving the set of problems provided by the educator, they will learn the required mathematics, and the educator expects the students to complete the problems through the given guidelines.

As the students work with the tasks, they are interacting with the milieu, which can be both collaborative and antagonistic. This period of interaction is the adidactical situation, where the students take the initiative and have the responsibility for the outcome of the learning process. An adidactical situation is the period ‘between the moment the student accepts the problem as if it were her own

and the moment when she produces her answer, [a time when] the teacher refrains from interfering and suggesting the knowledge that she wants to see appear' (Brousseau, 1997, p. 30). The adidactical situation persists for a time because the students know, through the didactical contract, that they would not have been set a problem they were unable to solve or learn from.

Preceding an adidactical situation is often a didactical situation, in which the educator or the task itself offers a problem to be solved. Succeeding the adidactical situation is another didactical situation, in which the educator or the task links the knowledge gained to the aim of the problem. Both the preceding and succeeding didactical situations do not depend solely on the educator, but can be parts of the designed task, here the mathematical programming tasks. Linking the knowledge gained to the aim of the problem is especially true of the succeeding didactical situation, in which the problem design can facilitate students' discussion of the implications of the solution of the task in a broader context. The educator's ability to recognise the students' actions and translate them into a system of what they ought to learn is important and is called the institutionalisation of the acquired knowledge (Brousseau, 2008). It is possible for the task to contain and facilitate institutionalisation through explanatory text or visuals connecting the actions to a system of learning. The task design should facilitate a situation whereby solving the task enables the students to gain the desired target knowledge.

Problem solving is a large part of mathematical thinking and learning and has long been a focus of mathematical research. According to Polya (1957), problem solving is an activity in which the subject or subjects goes or go through four phases, repeating phases when necessary. The four phases are orienting themselves in the problem, the creation of a plan, the execution of the plan, and the verification or checking phase (Polya, 1957). These phases can be repeated; if the execution or verification phase fails to reach a satisfactory solution, there will often be a return to the planning phase or even the orientation phase, as shown in figure 2.

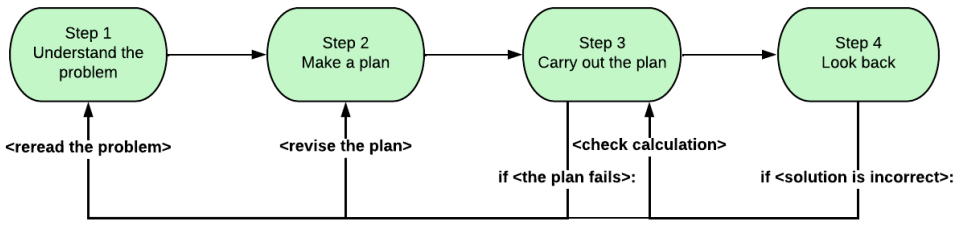


Figure 2: The four-step process for problem solving by Polya, altered slightly to fit the programming theme.

Polya's (1957) method of problem solving created the foundation for research into this field in mathematics education. Following Polya, A Schoenfeld (1985) gave a revised and broader sense of problem solving in mathematics, indicating that understanding and teaching in mathematics should be approached as a problem-solving domain. In Schoenfeld's (1985) work, four categories were required to achieve success in mathematics: resources, heuristics, control, and beliefs. Resources are the existing procedural knowledge in the student undertaking the task. Heuristics are strategies and techniques for working on mathematical problems, such as working backwards, drawing figures, and sorting information. Control is the decision making regarding which strategies to apply and when. Finally, beliefs are an overview of mathematics and will determine how a person approaches a problem.

During problem solving, students experience several different challenges, such as recalling previous knowledge, connecting previous knowledge to a current task, applying knowledge as execution, and verifying answers. Each part of the problem solving is accommodated to help with the building of the program. The ability to understand the problem, decompose it into smaller pieces, and solve each piece before reassembling them into a solution are particularly important parts of algorithmic thinking.

Knuth (1985, p. 170) defines algorithms as 'encompassing the whole range of concepts dealing with well-defined processes, including the structure of data that is being acted upon as well as the structure of the sequence of operations being performed '. Algorithmic thinking, meanwhile, is a type of mathematical reasoning (Stephens & Kadijevich, 2020) that can take a variety of forms, including, but not limited to, algebraic, geometric, and numerical. According to Stephens and Kadijevich (2020), algorithmic thinking is based on three cornerstones: decomposition, abstraction, and algorithmisation. Decomposition is about breaking the problem into smaller pieces to be solved individually and is a problem-solving strategy (Polya, 1957; A Schoenfeld, 1985). Abstraction refers to the process whereby learners attempt, succeed, or fail to reach an understanding of mathematical concepts, strategies, and procedures (Dreyfus, 2020) and is defined by Skemp (1986) as an activity by which we become aware of similarities. Algorithmisation reflects the process of converting a process into an algorithm.

The origin of algorithmic thinking links back to Seymore Papert's book Mindstorm: Children, computers, and powerful ideas (1980), in which the term 'computational thinking' was used to describe the thinking of children as they worked with LOGO programming to learn mathematics. The term computational thinking has not secured a footing in mathematics education and currently resides primarily within computational science, while retaining similarities to algorithmic thinking. Lockwood et al. (2016) link algorithmic thinking to the development of deep procedural knowledge, defined by Star (2005, p. 408) as 'knowledge of procedures that is associated with comprehension, flexibility, and critical judgement ', while Abramovich (2015) links it to the development of conceptual knowledge. Through the use of programming as a tool in the mathematics classroom, algorithmic thinking can be advantageous for mathematical learning, because decomposition, abstraction, and algorithmisation are also key ingredients applicable to many problems of mathematical content. Stephens (2018) points out that more 'attention to algorithmic thinking in schools could help students expand their problem-solving techniques and to explain and justify their mathematical reasoning' (pp. 489-490). The first hurdle to be overcome when applying programming in the mathematics classroom is that the students must learn to program.

3.2 Learning programming

How to learn programming is a large field (see e.g. Medeiros et al., 2018; Piteira & Costa, 2013). Much of the literature is focused on the introduction of programming at university level. Several studies show that many difficulties are encountered when introducing students to programming. PRIME focuses on the learning of programming syntax and sequences of variables, loops, and conditions as these are both the most applicable to a school setting and present a significant challenge to learners (Bosse & Gerosa, 2017).

The majority of school students are novice programmers and are limited to surface knowledge of programs, typically using the 'line by line' approach instead of meaningful programming building (Lahtinen et al., 2005). When learning programming at university, students often have knowledge of both the syntax and the semantics of programming, but they lack the ability to combine them into valid and efficient programs (Winslow, 1996). Although several attempts have been made to design strategies to support novice programming (such as collaborative teamwork, peer tutors, workshops, forums, etc.), multiple adversities still occur when students are introduced to algorithmic thinking (Stephens & Kadijevich, 2020), logic, and problem solving. Algorithmic thinking combined with the

exhausting labour of learning syntax generally leads to frustration, rejection, and poor visions of what programming is (Buitrago Flórez et al., 2017).

3.3 Adversities

Adversity is a term used in both Articles 2 and 3 and in PRIME. It is defined as any time students display uncertainty regarding how to proceed with the MPP on which they are working. It can take the form of questions to, or discussions with, the group through frustration, halting work for a period, to essentially giving up on the task. Adversity can be related to mathematics and/or programming. Barriers describe when the students encounter challenges in programming, while obstacles describe when they encounter challenges in mathematics.

Ko et al. (2004) present six learning barriers in end-user programming systems, of which the three that share properties with mathematics (selection, understanding, and information) are used in the current study. The remaining three (design, coordination, and use) are not used as they pertain to barriers only vaguely related to mathematics. The first of the barriers applied to this research, selection barriers, relates to the programming interface and knowledge of what tools and commands to use to build for a result. The second, understanding barriers, consists of knowledge of how to handle errors and unexpected behaviour of the program. Error handling is a recognised challenge within programming education (Lahtinen et al., 2005). The third, information barriers, refers to the program not confirming the result or behaving in accordance with the hypotheses, which often results in the inability to evaluate what went wrong. All three of these barriers are related to the evaluation of the program rather than its execution.

Ko et al. (2004) further divide the barriers into subgroups of surmountable and insurmountable barriers. Surmountable barriers are those students are able to overcome; when the complexity or number of barriers becomes too large, students are unable to continue and face an insurmountable barrier. The aim of the current research is for students to encounter surmountable barriers through design and exploratory talk, essentially enabling them to experience what TDS refers to as an epistemological obstacle. Similarly, insurmountable barriers are akin to what TDS calls ontogenic and didactical obstacles.

Making the students adapt strategies to reach the desired knowledge is challenging, and Brousseau (1997) suggests that they will not succeed unless the task forces them to do so. If the students encounter and overcome obstacles as they are working through the given tasks, adaptation may take place. Brousseau (1997) defines an epistemological obstacle as a form of knowledge that has been relevant

and successful in particular contexts, often including school contexts, but that at some moments becomes false or insufficient . Obstacles can also be of a different nature: ontogenic obstacles relate to limitations of the students and lack of required prior learning, and didactical obstacles relate to the presentation of the subject, or 'the result of narrow or faulty instruction' (Harel & Sowder, 2005, p. 34). Both ontogenic and didactical obstacles inhibit learning and should be avoided (Brousseau, 1997; Harel & Sowder, 2005), while epistemological obstacles should not be avoided, but considered to be an important part of a good task design. An epistemological obstacle is explained by Balacheff (1990, p. 264) as follows:

> Any content has to be supported by the pupils' previous knowledge. But this old knowledge can turn into an obstacle to the constitution of new conceptions, even though it is a necessary foundation. But more often than not, to overcome this obstacle is part of the construction of the meaning of the new piece. (p. 264)

This explanation indicates that task design needs to build on previous knowledge to both create new conceptions and an adidactical situation in the design process to facilitate the construction of new meaning, which links back to how 'the construction of a web of connection' (Noss & Hoyles, 1996, p. 105) facilitates mathematical learning. When students are working on tasks, they will experience periods when they know how to perform the required actions and periods when they encounter obstacles. These adidactical situations will allow the students to reconsider their strategies, develop new pathways, discuss with their peers, conjecture, and experiment, all of which are related to the intended learning process (Leung & Baccaglini-Frank, 2016). Thus, adidactical situations have the potential to stimulate mathematical thinking and reasoning and to develop conceptual knowledge (Hiebert, 2013; Skemp, 1976), leading to the acquisition of deep learning of the required mathematical content (Noss & Hoyles, 1996). Obstacles are closely related to the struggle defined by Hiebert and(Hiebert & Grouws, 2007), in which making sense of mathematics takes centre stage.

It is important to separate what constitutes a positive adversity from what constitutes a negative or undesirable one. Epistemological obstacles and surmountable barriers are adversities which allow students, through exploratory talk, to solve the problem. The solution results in combining elements of existing knowledge to form previously unknown knowledge. This is a positive adversity. Ontogenic and didactical obstacles and insurmountable barriers are adversities

which leave students displaying frustration and a general sense of negative premonition (a 'this is not going to work' mentality), often leading to their giving up on the task, thus being a negative adversity.

3.4 Task design

In PRIME, tasks are defined as information that prompts student work, presented to them as questions and instructions that are both a starting point and context for their learning (Watson & Sullivan, 2008). As tasks are the main component of the 'things to do' in the mathematical classroom (Watson et al., 2015), combining mathematics and programming in a design task is a valuable endeavour. Moreover, I argue that engaging the students in tasks where they both think about mathematics themselves and participate in a mathematical discussion with their peers is the main stimulus for learning (Anthony & Walshaw, 2009; Sullivan et al., 2012). This section presents task design in mathematics combined with programming, before describing the elements of didactical engineering (DE) used in the design of this work.

The design of mathematical tasks can be divided into several categories, ranging from open- versus close-ended tasks through contextualised versus non-contextualised tasks to routine-based versus cognitively demanding tasks (Berisha & Bytyqi, 2020; Stein et al., 1996). Stein et al. (1996, p. 426) state that 'tasks used in mathematics classrooms highly influence the kinds of thinking processes in which students engage, which, in turn, influences student learning outcome' (p. 426). Designing tasks that use digital technologies is complex and difficult (Joubert, 2007; Laborde & Sträßer, 2010), partly because of the computer's ability to perform hidden or unknown mathematical procedures (referred to as 'black box' by Buchberger (1990). In comparison, a 'white' (or transparent) box is characterised by making the students conscious of the mathematics they ask the tool to perform. Rabardel (2002) also uses the concepts of 'black box' and 'glass box' to refer to the dimension of the operative transparency of the technological tool being used.

A review of research into computer education strongly indicates that learning to program is difficult as 'students exhibit various misconceptions and other difficulties in syntactic knowledge, conceptual knowledge, and strategic knowledge' (Qian & Lehman, 2017, p. 17). Syntactic knowledge is the understanding of the building of the code, such as the use of parentheses, equals signs, and semicolons. Conceptual knowledge is the students' model of code executions, such as variables, if statements, and the sequential execution of the code. Strategic knowledge is the process of planning, writing, and debugging programs. Moreover, text-based

programming creates additional challenges for students, such as focusing on syntax rather than the mathematical meaning of the code (Lewis, 2010; Resnick et al., 2009).

Ideally, when tasks are being worked on, the milieu should 'provide feedback that moves the learner forward' (Wiliam & Thompson, 2008, p. 15). Feedback can come from the task, the group conversations, or the program. The tasks facilitate learning by generating elements and accompanying actions that the students undertake together with the feedback provided by the milieu. Feedback should enable the students to evaluate meaningful strategies, which attest to the building of new knowledge (Artigue et al., 2014). In task design, including technological tools, feedback can be particularly important (Bokhove and Drijvers (2010). The feedback helps the students to construct knowledge as they become engaged in the solving of the problem and refine their concepts and strategies (Brousseau, 1997). Such cognitive development, in the TDS framework, is part of the adidactical situation.

A property of technological tools is, to varying degrees, the provision of valuable feedback to the user. Feedback from a computer is 'quick and essentially unlimited… at "no cost"' (Hillel, 1992, p. 209), allowing task designers to facilitate a greater range of experimentation with and verification of ideas. The ability to use technology to produce results, immediate feedback, and novel ways of looking at mathematical objects could support a change in the nature of communication in mathematical problem solving (Drijvers et al., 2016). The change creates a challenge for designers, as tasks ideally allow for experimentation, exploration, and discussion. Several studies present guidelines, criteria, and/or principles for designing good mathematical problems (e.g. Johnson et al., 2017; Kieran, 2019; Lithner, 2017; Sullivan et al., 2012; Wiliam & Thompson, 2008), advocating a focus on conceptual, open-ended tasks, linking technical and theoretical activity, integrating questions calling upon pattern seeking, and applying technological tools for generating and testing conjectures.

One of the goals of the design of mathematical tasks utilising programming is to enable the students to make new connections between concepts and create new concepts. This mathematical learning will then cause an increase in mathematical understanding. Conceptual blending describes how new connections can occur when two representations or concepts are brought together in a 'blended' concept (Fauconnier & Turner, 2008). This mixing can be thought of as a combination of two or more concepts into a third, newly formed, mental space (Whiteley & Mamolo, 2013). In order to form this new concept, the task has to offer the opportunity to do so; that is, it must give meaning to the newly formed concepts (De Lange, 1987). The

newly formed concept or network between concepts has created a conceptual change in the students (Vosniadou, 1994, 2003). This change is often characterised as difficult for students as it conflicts, in varying degrees of strength, with their previous knowledge. As an example, the conceptual change from rational numbers to irrational numbers is less demanding than a change from real numbers to imaginary numbers (Lehtinen et al., 1997).

Both conceptual 'blending' and conceptual change are applied in the creation of the tasks as the students investigate numerical methods in mathematics by weaving together their previous mathematical knowledge. When the students' experiences conflict, whether through weaving together previous concepts to a new concept or building connections between concepts, they will experience an obstacle. This period is important for the design of the tasks and results in a suggestion being made by a single student or in collaboration with the rest of the group. The task is then to facilitate an adidactical situation in which the students are building new knowledge. The task could facilitate mathematical learning and understanding, using elements of problem solving and positive adversities to initiate group discussion. At the same time, it could mitigate negative adversities. The process of the design of tasks is explained and exemplified in the section addressing MPPs in the methods chapter.

4 Methods

The methods chapter departs from the goal of designing problems in which programming is combined with mathematics to facilitate mathematical learning. The chapter initially presents the DBR of PRIME together with an argument for this choice of methodology. It was decided to present DBR before describing the design of the MPPs as it illustrates the background to the design process. The next section presents, in detail, the process of designing the MPPs together with the iterative process involved in the design process. Thereafter, the context, elements of data collection, and framework used for the different analyses in PRIME are presented. The range of ethical considerations influencing PRIME is wide, and the final section in this chapter presents the different challenges encountered throughout this work.

A schematic presentation of the methods chapter is presented in figure 3. The MPPs are the centre piece of PRIME and therefore affect every other element of the method. The MPPs are explained in detail in 4.3. The bottom row of figure 3 builds towards the research question, presenting the broad structure of the research before narrowing down to data collection and analysis. Finally, ethical considerations are discussed.

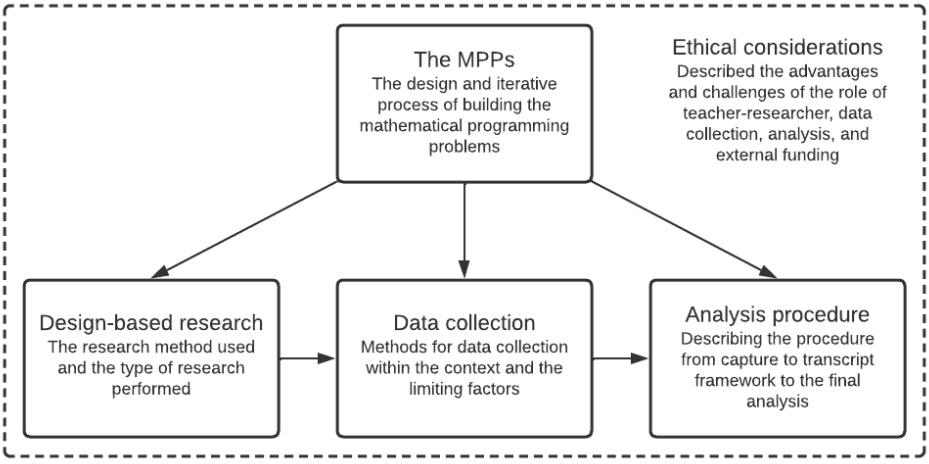


Figure 3: Schematic overview of the method chapter.

4.1 Research design

The overarching design of PRIME is the implementation of MPPs in a classroom. Figure 4 shows how it was implemented throughout the first three years.

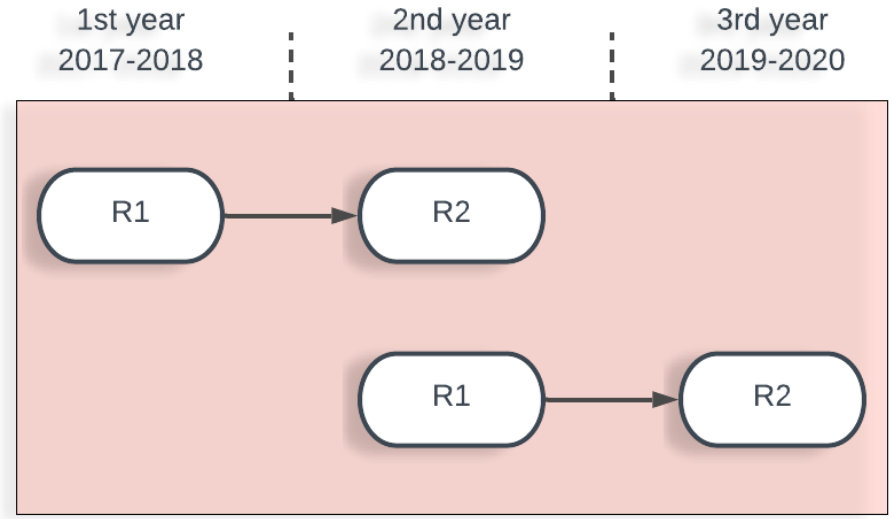


Figure 1: The implementation of programming in mathematics classrooms. R1 and R2 are elective science classes in Norway.

DBR in mathematics education is explained by Swan (2013) as 'a formative approach to research, in which a product or process (or "tool") is envisaged, designed, developed, and refined through cycles of enactment, observation, analysis and redesign, with systematic feedback from end-users' (p. 192). A variety of tools is used, from innovative teaching methods through didactic materials, teacher training, and assessments. Thereafter, 'educational theory is used to inform the design and refinement of the tools and is itself refined during the research process' in order to 'create innovative tools for others to use, to describe and explain how these tools function' (p. 192). Furthermore, theory can explain why a given range of implementations occurs and 'develop principles and theories that may guide future designs' (p. 192). The goal is transformative, seeking to create new teaching and learning possibilities and study their impact on students.

This description indicates several similarities between DBR and DE (Artigue, 2015). The current research uses Anderson and Shattuck (2012) definition of DBR as it is largely applicable to PRIME. First, the research took place in a real educational context and was conducted by a teacher/researcher. The implementation of PRIME occurred in a classroom, and the implementation was

conducted by the writer, as a teacher–researcher. Second, the focus was on the design and testing of a significant intervention (i.e., the implementation of programming). However, the current research deviates from the third point in DBR as defined by Anderson and Shattuck (2012), that mixed methods are used, as it is much more closely related to qualitative than quantitative methods. Although it uses elements of quantitative methods, considering the small number of participants (N = 58), PRIME does not qualify as mixed-methods research.

The fourth element in Anderson and Shattuck's (2012) definition is that DBR involves multiple iterations. As described in section 4.2, there are several iterations to both the design and the implementation of the mathematical programming tasks. As regards the fifth point, it involved a collaborative partnership between researchers and practitioners: in both the design and implementation, there were discussions with both researchers and teachers from which the design benefitted. Finally, as regards the sixth point in the chosen definition, that DBR should involve the evolution of design principles, although PRIME does not provide such principles, it does provide design recommendations. Since programming in the mathematics classroom in upper secondary school is still in its infancy, it would be problematic to use the word 'principles' until more research has been conducted. The recommendations do, however, present a starting point for the work towards design principles.

The first three years of PRIME can be viewed as research undertaken by a practitioner in order to improve or develop their practice (Brydon-Miller et al., 2003; Corey, 1954). After the combined task design and implementation process, the last part of PRIME started the process of developing principles for designing MPPs in the classroom, whereby PRIME evolved from a local initiative into an intervention for a more general classroom (Brown, 1992). DBR contains a 'meta' paradigm, pragmatism, where the truth lies in the utility (Cole et al., 2005). In the current case, the utility is contained within the design recommendations. DBR aims to 'not only meet local needs, but to advance a theoretical agenda, to uncover, explore, and confirm theoretical relationships' (Barab & Squire, 2004, p. 5).

Additionally, DBR may involve more than simply describing the design and the conditions under which it changed. Cobb et al. (2003, p. 10) suggest that 'design experiments are conducted to develop theories, not merely to empirically tune "what works"'. As an example, Barab et al. (2000) conducted DBR in a university classroom context, iteratively refining course materials each semester so as to advance a participatory learning framework that was conceptually rich and theoretically grounded. The continued development through an iterative process is

positive in that there is always room for improvement and negative in that it is difficult to decide when the project is finished (Anderson & Shattuck, 2012).

The aim of the research question is to provide recommendations guiding the design of tasks in which programming is combined with mathematics to facilitate mathematical learning. These recommendations will reflect the environment in which they are developed (Anderson & Shattuck, 2012), and through several iterations the environment will incorporate several mathematics students. This aim was achieved though the three years of data collection. Further, as (Barab & Squire, 2004, pp. 5-6) state:

> Although providing credible evidence for local gains as a result of a particular design may be necessary, it is not sufficient. Design-based research requires more than simply showing a particular design works but demands that the researcher (moves beyond a particular design exemplar to) generate evidence-based claims about learning that address contemporary theoretical issues and further the theoretical knowledge of the field. (pp. 5-6)

DBR is sometimes characterised as a form of DE (Artigue, 2015; Godino et al., 2013), where something must be made with whatever theories and resources are available. The products of DBR are judged on innovativeness and usefulness, not just on the rigor of the research process, which is more prominent in evaluating true experiments (Plomp, 2013).

DE is one approach to creating and evaluating the design of tasks and its obstacles in a controlled way (Brousseau, 2008). DE consists of several phases (Artigue, 2015): (1) a preliminary analysis of the mathematical content, possible constraints, and existing research. Where the implementation of the mathematical content in schools is concerned, a review of previous research into task design and mathematical learning and the evaluation of possible outcomes scenarios are central; (2) a design and an a priori analysis, which determine and evaluate the didactic variables influencing the interaction between the mathematical content and the milieu, consisting of the student, the task, and the programming language; (3) the realisation, observation, and collection of data which enable an understanding of students' interaction with the milieu, how the task design facilitates their mathematical learning, and what adversities they encounter; (4) an a posteriori analysis and validation, which is a review process concerning all the previous phases, contrasting with the a priori analysis. Evaluation of the dynamics within the

milieu and performing alterations to the initial design are part of the a posteriori analysis and was performed several times during the design iterations.

The methodology of DE aims at creating situations in which the acquired mathematical knowledge provides a successful solution to the given problem, reached by the students through interaction with the milieu. The role of the educator is to facilitate devolution, in which the students accept the mathematical responsibility of solving the problem and thereby develop an adidactic interaction with the milieu. A full preliminary analysis was not performed, as the initial design was based on the personal experiences of the teacher–researcher. Article 1 investigates the design, an a priori analysis, and the realisation of the design. Articles 2 and 3 investigate the realisation in the form of implementation and the accompanying observation, as well as data collection and analysis.

The design is analysed through evaluating the didactic variables influencing the interaction between the mathematical content and the milieu. The didactic variables consist of linking programming as a tool to learning mathematics, considering the added difficulty that programming brings to mathematics, enabling students to discuss mathematics and facilitating mathematical learning. As Artigue (2015) observes, 'these variables condition the milieu, thus the interactions between students and knowledge, the interactions between students and between students and teachers, thus the exact opportunities that students have to learn, how and what they can learn' (Artigue, 2015, p. 5).

Iteration is characteristic of DBR and creates the foundation for the task design process. The iteration process of the tasks consists of several steps. It starts with an idea of a possible coalescence of programming and mathematics and, after a creation period, ends up with a sequence of MPPs for a lesson. As is not uncommon in DBR, the initial build was a result of researchers taking their best bets (Lehrer, 2001), meaning that some aspects of the design had to be changed at a later date. At the same time, in order to optimise the learning environment, many researchers, including myself, are involved with teaching (McClain & Cobb, 2001; Smit & Van Eerde, 2011). A description of the creation process is presented in figure 3.2, with a more detailed description in Article 1.

As PRIME explored relatively new territory, the choice for data collection was broad at the start. Initially, video recordings, group interviews, student and teacher logs, and anonymous surveys were used to gather information about the task design and implementation. It quickly became apparent that only some of these methods for collecting data were of value at this stage. The chosen method was video recording the students' computer screen together with voice recordings. Given this

form of data collection, PRIME stands firmly in the field of qualitative research. As it is argued that some quantification from qualitative research can uncover the generality of phenomena being described (Silverman, 1985), Articles 2 and 3 include elements of quantitative research. The inclusion of quantitative data also eliminated the use of terms such as 'many' , 'some', and 'often' (Bryman, 2016) in favour of precise numbers. However, since the data were primarily collected through transcription and both Articles 2 and 3 present the frequency of the different codes in the coding scheme, this does not make it a quantitative procedure (Gerbic & Stacey, 2005).

4.2 The design of mathematical programming problems

With the ambition of facilitating mathematical learning (Noss & Hoyles, 1996) through algorithmic thinking (Stephens & Kadijevich, 2020), the MPPs in PRIME were designed based on the belief that the adidactical situation (consisting of the milieu, feedback from the milieu, and the didactical contract between students and educator (Brousseau, 1997)) constitutes the most important part of the learning process.

Several factors were considered in the decision about which programming language to use. The programming language had to have a low floor and a high ceiling, where a low floor means that the language is easy to learn and use, and a high ceiling means that the language can be used for a wide range of problems and situations. The programming language and at least one of its editors had to be freely available so that the students could install and run the programs on both their school computers and home computers. The editor also had to have a simple interface to prevent the students from encountering obstacles outside mathematics and programming. Finally, the languages used in previous research and both school and university courses were reviewed. Many excellent programming languages, such as Matlab, while fulfilling many of these criteria, are not free of charge and therefore not eligible for use in a school setting. From the above criteria, two languages were considered, namely Scratch and Python.

Scratch, developed by MIT Media Labs in 2005, is aimed at pupils, is both easy to learn and visually interactive, and has been used in mathematics education in several studies (Han et al., 2016; Lee, 2011; Sáez-López et al., 2016). Scratch is a block-based programming language in which the user builds a program by assembling block. Each block constitutes one action; for instance blocks of 'move 100 steps' and 'turn left 90 degrees' can be assembled to create a square like LOGO.

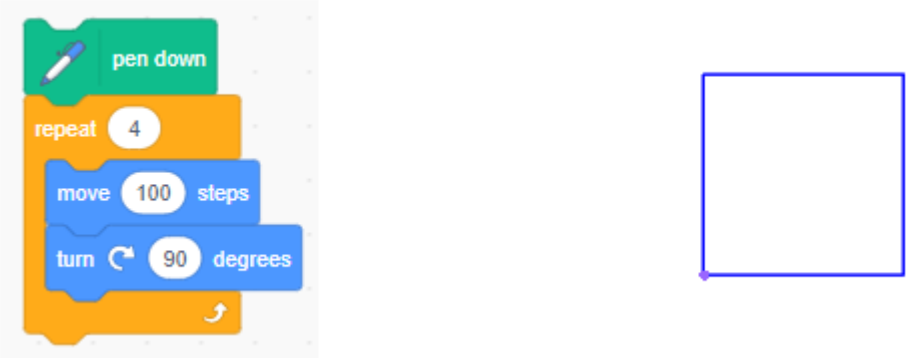


Figure 5: Constructing a square in Scratch.

Much prior research used a block-based programming language such as Scratch, but this is partly due to the fact that programming in lower school has primarily been the object of investigation. Scratch was a consideration, but after initial trials the students were observed to find it 'too childish' and did not view it as a 'proper' programming language. The research also indicates some concern that learning block-based programming creates possible challenges when students later learn a text-based language (Moors et al., 2018; Weintrop et al., 2017). In upper secondary school, the text-based programming language Python was considered a better fit. This selection also conformed to the trends in both the curriculum and textbooks in Norway, whereby block-based programming languages are implemented in primary school, and the shift to text-based programming is undertaken in secondary school.

Python is considered by many to be a good language with which to start learning programming (Pajankar, 2017; Payne, 2017; Wainer & Xavier, 2018). The choice of Python as the text-based programming language in PRIME was multifaceted. It is easy to start learning to code with Python; for instance, there is no need to define variables prior to using them, as in some other programming languages. This characteristic is often referred to as a low floor (Papert, 1980), indicating a low threshold for learning. The syntax structure of Python is also like the English language, making it easy to read code. Python is a widely used programming language amongst both universities (Mason et al., 2018) and large corporations, such as Google and NASA, which demonstrates the wide selection of applications possible within it. This characteristic is often referred to as a high ceiling (Papert, 1980). Python is also open source, which, in a school setting, is important so there is no extra cost to the institution of implementing Python into

their system and the students can install it on their home computers free of charge. As of writing (spring of 2022), Python is the language used in upper secondary schools in Norway.

During PRIME, the students have used three different editors, as they developed new features over time[1]. The iterative design process was applied throughout the work in PRIME and was called 'revisions'. Figure 6 shows the three types of revisions that took place during PRIME. Immediate revisions were those that happened within a year and within a class, affecting future MPPs for the same class later in the same year. Progressive revisions were those made for the following year with the same group. These were made because each group is different, and the changes made to each separate group needed their own revisions. They were not significant changes, but could depend on factors such as special interests within the group. As an example, the first progressive revision changed because the students became interested in fractals, and building a program visualising different fractal patterns was implemented in their final year. Annual revisions were the largest change, whereby all the data collected, logs, and experiences from the previous year were fed into the next iteration of the MPPs.

[1] The first editor was PyCharm, followed by Spyder, which was used for most of the research period. At the end of the third year, the research migrated to Jupyter Notebook, which allowed for a more user-friendly interface for the students and made sharing with other teachers easier. The usage of Jupyter could have its own section, but since it was implemented at a very late stage, it was not part of the research.

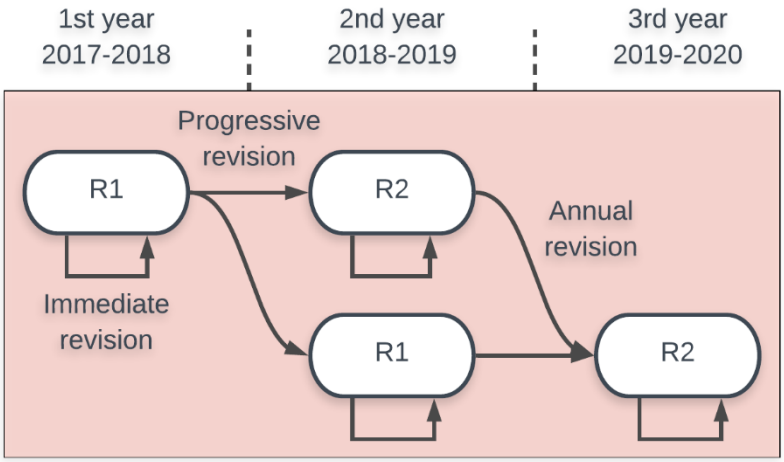


Figure 6: Structure of the iterative cycle for revisions throughout the implementation.

A pilot set of MPPs was designed through the teacher knowledge possessed by the researcher. As no research within the field of mathematics presented how to implement programming in upper secondary school mathematics, the idea of the pilot was to explore properties of mathematics with the use of programming. With this mindset, a few MPPs were designed covering relatively simple methods, such as solving quadratic equations and a set of linear equations. These MPPs were very simple and consisted primarily of a short description of what the program should do. An example is 'create a program that solves a second-degree equation and prints the possible solutions to screen'. The students were given this task early in the first year but after they had learned some basic programming commands and structures, such as if statements and plot commands. As the students worked on the pilot MPP, several groups were filmed. These films, together with the researcher's notes from the lesson, were used to revise and build the next iteration of the MPPs.

The initial MPP had too little information regarding how to program, and the students encountered barriers and asked questions regarding programming code instead of mathematics. The first major revision redesigned the MPP to include more assistance with programming as the students worked through the task to alleviate some of the barriers. As an example, more skeletal code was used as this enabled the students to focus on mathematics rather than programming. Figure 7 presents a typical example of a skeletal code in which the structure of an if

statement is written, but several of the conditions, calculations, and outputs are missing.

```
if d > 0:
    x1 = insert_calculation_here
    x2 = insert_calculation_here
    print("There are two zero-points located at x =", x1, "and x =", x2)
elif insert_condition_here:
    insert_calculation_here
    print("insert_output_to_user")
else:
    print("insert_output_to_user")
```

Figure 7: Skeletal code given to the students. The programming structure is given, but the mathematics must be implemented.

After viewing the recordings and comparing the students' actions and interactions with the researcher's logs from the initial run of the MPP, the next iteration of the design was built. The revised design consisted of four parts:

I. The MPP started with the students recollecting previous concepts and combinations related to the mathematical program they were about to build. The process of recollection facilitated the presence of the necessary mathematical concepts in the students' minds and the design of a set of initial tasks that all the students could answer. The aim was for this recollection to build motivation and, possibly, stamina for the remainder of the MPP.

II. The MPP then asked the students to explore the limits and boundaries of the program they were about to create, including asking questions such as 'When does this procedure not work?' and 'What special cases do we need to consider when solving?'. A simple example is quadratic equations, for which the students needed to discuss and differentiate between cases with two solutions, one solution, and no solutions.

III. The next part addressed how to resolve the limits and boundaries uncovered in the second part, where the MPP moved closer towards programming. This section aimed to build a strategy for solving for all possible outcomes (Polya, 1957; A Schoenfeld, 1985). As an example, the following question was posed: how can you check for no solutions of a quadratic equation before using the quadratic formula? In addition to learning how to check for boundary issues, the students learned algorithmic thinking, which refers to the sequence in which operations

are needed. The sequence of algorithms in the example presented is to first calculate the value of the discriminator and, if the discriminator is equal to or greater than zero, to then apply the quadratic equation.

IV. Finally, the MPP introduced programming and assisted the students to structure a program by solving or visualising the mathematical procedure or concept of the previous parts. The MPP strove throughout to facilitate the students' engagement in mathematical discussions and challenges. The engagement was through task design and could vary in difficulty, but the more difficult tasks were always presented towards the end of an MPP so as not to hamper the students in their progress towards building the program. As mentioned previously, a skeletal code (figure 7) was often given to the students, as the subject was mathematics, and programming was viewed as a tool for understanding mathematics.

These four points did not always occur in sequence; the last point, in particular, could be intermingled with the previous three if needed. After the second iteration, the MPP was again implemented in a mathematics classroom, and data were collected through video recordings, interviews, and logs from the lessons. The data were viewed and analysed to build the next iteration of the MPP.

The final design structure consisted of two parts. The first part consisted of non-programming tasks asking the students to recall previously known mathematical concepts. There are two main reasons for this. First, it ensured that all students had the opportunity to recall the same information, which was applicable and beneficial (Stillman, 2004) throughout the problem. Second, it had a low threshold for completion and facilitated the students' ability to complete the initial part of the task. This characteristic upholds the part of the didactical contract between students and educators whereby the students expect to be able to resolve the problem with their knowledge. Later, when the complexity increased, the recollected knowledge facilitated the students in solving the problem. The problems continuously built upon the recalled knowledge to facilitate mathematical learning and attain the intended knowledge. The MPP can be viewed as a process in which each new problem facilitated an adidactical situation, aiding the students in their progression towards the desired knowledge (Brousseau, 1997).

Problems can also directly or indirectly reveal inadequacies or challenges within mathematics that can, to a lesser or greater extent, be solved through programming. The programming part started in a similar way, recalling a set of previously known programming procedures necessary to build the required

program. As the students worked through the different problems of the MPP, they were continuously changing and developing the program. Each section of code that it was possible to execute represents a verifiable step towards the desired knowledge. Verification can be carried out by inspecting graphs, output data, or some other form of visual or readable response. The various sections of the problem design facilitated an increased transparency of the problem, in which the students became conscious of and engaged in the mathematical knowledge that they applied through the programming process. Thus, a balance was created between the additional complexity programming brings to the mathematics classroom and the overall design of the problem. The complexity, in terms of the lower versus higher level of demands of the problem (Stein & Smith, 1998), together with the students' previous knowledge and self-efficacy, determined whether they would succeed or not in solving the given problem (Hoffman & Spatariu, 2008). The maintenance of high-level (cognitive) demands is dependent on several factors, including thinking and reasoning, self-monitoring, building on previous knowledge (Stein & Smith, 1998), and the link between the problem and the aim throughout the MPP. Soloway (1993) argues that during the programming process, the learner uses powerful problem- solving and thinking strategies. Students first need to solve a problem mathematically, then reflect on how to express the solution through computer programming (Papert, 1980; Szlávi & Zsakó, 2006).

A representation of the design structure (as presented in Article 1), emphasising the adidactical situations in the MPP, is presented in figure 8.

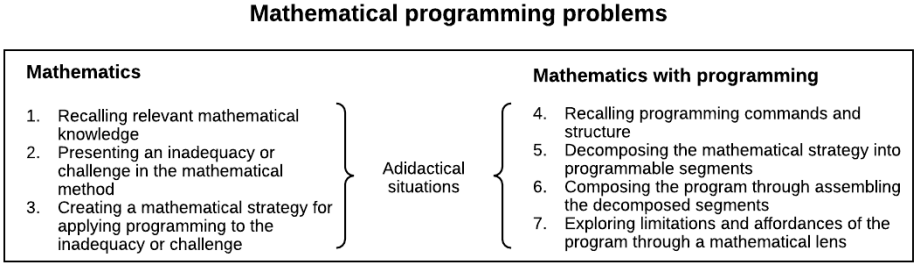


Figure 8: A representation of the design ideas of the MPP.

To exemplify the design structure, an excerpt of an MPP as presented in Article 1 is given below.

1. **Recalling relevant mathematical knowledge**

   Enabling the students to recall the mathematical knowledge required to initiate the progress through the MPP and their search for the target knowledge. This can limit the amount of ontogenic obstacles.

   In figure 9, this consists of the calculation and discussion of the properties of zero point for different functions, which allowed the students to recollect the properties of the zero point and method for finding zero points, which was the base for the target knowledge.

Discuss with your group how (and if) one can find the zero-points for all the function-types below.

a) All quadratic equations, for instance

$$f(x) = 2x^2 + 3x - 2$$

b) All third-degree polynomials, for instance

$$g(x) = x^3 + 3x - 4$$

c) All rational functions, for instance

$$h(x) = \frac{2x + 3}{3 - x}$$

d) All exponential functions, for instance

$$i(x) = e^{2x+1} - 1$$

What are the differences and similarities between the method used to find the zero-points in these functions?

Figure 9: Problem from phase 1 of the MPP concerning the bisectional method.

2. **Presenting an inadequacy or challenge in the mathematical method.**

   The start of a problem presented the students with one or more examples of when a mathematical method becomes false or inadequate. This forced the students to search for new strategies, as they, through the didactical contract, were confident that a solution and learning opportunity existed.

   In figure 10, this consists of the search to find zero points for a general and unknown continuous function covering a range of both negative and positive function values. This problem allowed the students to use and

apply their knowledge of zero points to a new type of problem. The strategy they started to develop here was used later in the same MPP.

Imagine a machine that contains and unknown linear function and will give you a function value when you input a numerical x-value. All non-numerical inputs will yield an error.

How can you systematically use this machine to find the zero-point for the unknown function to the right? What if the function is non-linear?

The machine can be as accurate as you would like concerning the function value. Discuss possible strategies with your group.

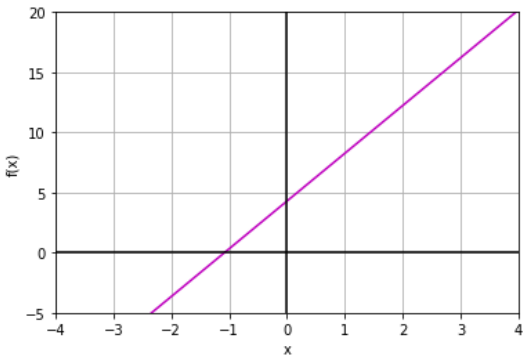


Figure 10: Problem from phase 2 of the MPP concerning the bisectional method.

3. Creating a mathematical strategy to apply programming to the inadequacy or challenge

Selected problems initiate a process of combining existing mathematical knowledge with new strategies or concepts. This further builds towards the target knowledge where the students, through the adidactical situation, can develop new pathways through interaction with the milieu. Phase 3 was designed to facilitate exploration, discussion, and evaluation, allowing the students to assemble their existing knowledge into a new strategy or method.

The problem in figure 11 consists of developing strategies to numerically approximate the zero points through interacting with the milieu. The problem facilitated the students discussing and testing different strategies, building towards the target knowledge.

Elect one group member to take the role of the machine from the previous problem. The «machine»-students collects a set of graphs from the teacher, where the zero-point is marked with an accuracy of six decimal places. The «machine»-students can only reply with function-values, when given x-values from the other group members. The task is to find the zero-point(s) of the unknown graphs as efficiently as possible. You may discuss strategies with everyone in the group, including the «machine»-student.

Figure 11: Problem from phase 3 of the MPP concerning the bisectional method.

4. Recalling programming commands and structures

Enabling the students to recall the programming knowledge required to initiate the progress through the MPP and their search for the target knowledge can limit the number of ontogenic obstacles.

In the presented MPP, this consists of plotting, implementing elementary calculations, the use of functions, and the structure of while loops. Figure 12 is a typical code written by the students to recall how to plot (lines 10–17) and how to build (lines 3–5) and call (line 8) a function. The result after running the code is shown in figure 13.

```
1   from pylab import *                 # Importing required commands
2
3   def function_value(x):              # Function calculating the y-values
4       y = 2*log(x**4 + 4) - 0.5*x
5       return y
6
7   x = linspace(-5, 80, 1000)          # Values for x
8   y = function_value(x)               # Calling the function above
9
10  plot(x, y, 'g')                     # Plotting x and y
11  grid()                              # Draw a  grid
12  ylim(-5, 15)                        # Sets the y-interval
13  xlabel('x')                         # Labels the x-axis
14  ylabel('f(x)')                      # Labels the y-axis
15  axhline(y = 0, color = 'k')         # Draws the x-axis
16  axhline(y = 0, color = 'k')         # Draws the y-axis
17  show()                              # Shows the graph
```

Figure 12: Code snippet from phase 4 of the MPP concerning the bisectional method.

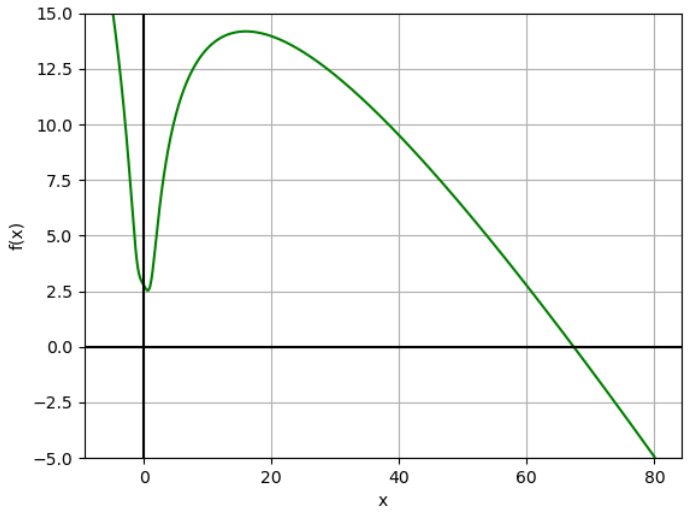


Figure 13: Result from running the code in figure 12.

5. **Decomposing the mathematical strategy into programmable segments**

Using programming as a tool, the students implemented mathematical strategies to create a basic program facilitating explorative search and analysis. Programming facilitated algorithmic thinking to decompose the mathematical strategy into smaller programming sections and implement it in the program.

In the presented MPP, this consists of decomposing the strategy into smaller, programmable mathematical segments. As an example, the students needed to input two starting x-values (xl and xr) where the function values (yl and yr) had different signs for the program test for the middle value and initiate the numerical procedure (figure 14 and figure 15).

```
20  xl = float(input("x-value left of the zero-point: "))
21  xr = float(input("x-value right of the zero-point: "))
22
23  yl = function_value(xl)                  # Function value, left side
24  yr = function_value(xr)                  # Function value, right side
25
26  print("Left side: f(", xl, ") =", yl)    # Prints the function value
27  print("Right side: f(", xr, ") =", yr)   # Prints the function value
```

Figure 14: Code-snippet from phase 5 of the MPP concerning the bisectional method.

```
x-value left of the zero-point: 60

x-value right of the zero-point: 80
Left side: f( 60.0 ) = 2.754757115060663
Right side: f( 80.0 ) = -4.943786727296455
```

Figure 15: Result from running the code in figure 14, where the inputs are 60 and 80.

After calculating the middle value (xm) between the two x values and the corresponding function value (ym), another part of the decomposed strategy was to build a segment that checked the sign of the function value (figure 16 and 17).

What is a quick way to check if two numbers have the same or opposite signs with the help of an *if*-statement? Discuss with your group.

Figure 16: Problem from phase 5 of the MPP concerning the bisectional method.

```
32      if yl * ym < 0:
33          # Possibility 1: Zero-point located between xl and xm
34      elif yr * ym < 0:
35          # Possibility 2: Zero-point located between xm and xr
```

Figure 17: Code snippet and example of a skeletal code solution to problem in figure 16.

6. **Composing the program through assembling the decomposed segments.**

   The students built the program by applying algorithmic thinking to the decomposed segments of the mathematical strategy, resulting in both abstraction and algorithmisation. The composing of the program implemented the mathematical strategy to resolve the inadequacy or challenge.

   In figure 18, this consists of combining all programming segments into a completed code. The part of this completed code shown below illustrates a while loop numerically calculating the zero point. This code is built upon the code in figure 17.

```
39      while abs(xr - xl) > 0.001: # Set accuracy of zero-point value
40          if yl * ym < 0:
41                  # Possibility 1: Zero-point located between xl and xm
42                  xr = xm                     # Updates right x-value
43                  yr = function_value(xr)     # New right function-value
44          elif yr * ym < 0:
45                  # Possibility 2: Zero-point located between xr and xm
46                  xl = xm                     # Updates left x-value
47                  yl = function_value (xl)    # New left function-value
48
49          xm = (xv + xh) / 2                  # New middle value
50          ym = function_value (xm)            # New middle function-value
51
52      print("The zero-point is x =", round(xm, 2))
```

Figure 18: Code snippet from phase 6, where several elements (from figure 12 and figure 14 (lines 43 and 47), figure 17 (lines 40 and 44)) are combined into a code segment.

7. **Exploring limitations and affordances of the program through a mathematical lens.**

After the creation of a program, there is a need for evaluation. Revising and discussing the affordances of the program allowed for consolidation of the knowledge throughout the MPP. Investigating the limitations of the program facilitated open-ended mathematical exploration and discussion.

In figure 19, this consists of discussing the limitations of what functions the numerical bisectional method applies to.

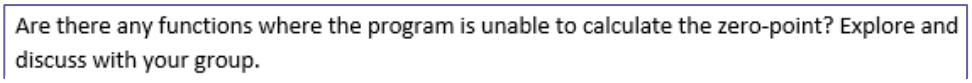


Figure 19: Problem from phase 7 of the MPP concerning the bisectional method.

For a complete view of the entire MPP as given to the students in Norwegian, see appendix. Since I have now presented and defined that a task in which programming is combined with mathematics is an MPP, the research question is:

What recommendations should guide the design of MPPs to facilitate mathematical learning in upper secondary school?

4.3 Context, participants, and data collection

The participating students had all elected science mathematics in the second year of upper secondary school in Norway and specifically chosen to be a part of PRIME. Prior to selecting their elective subjects, they were given information regarding this class and PRIME and had the option to participate or not. There were several science mathematics classes at the school, so they did not have to join if they did not want to. This situation creates two validation issues for this work. Science students are often more motivated than non-science students (Andersen & Cross, 2014) and have greater stamina and tolerance for a higher workload. Additionally, the students elected to be a part of PRIME specifically, so the class was not a representative group of a general mathematics classroom. Initially, there was a concern that this would reduce the validity of the research due to a reduced number of participants and that only either high-achieving students or students with prior knowledge of programming would participate. Reviewing the students' mathematics grades from the previous year together with an anonymous survey asking the students for their knowledge of programming revealed that neither of those concerns were present.

Two classes were selected for PRIME and investigated over two years. The first class, of second-year science mathematics students ('R1') started in the autumn of

2017. This group of 27 students was part of the initial pilot study, and the data collection started after only a few weeks of term. The following year, the same group, now in its final year of science mathematics ('R2'), was the subject of PRIME. At the same time, PRIME started with a new class in the year below and similarly followed this class for two years, as illustrated in figure 6. The students worked on the MPPs in groups of two to four to facilitate the adidactical situation and encourage exploratory talk. Each group worked together to solve an MPP, but each student created their own program.

The students discussed the MPPs through the process of problem solving, whether it was building concepts, building networks, specific commands, order of operations, possible solutions, and so forth. The group discussions and their coding were analysed for this thesis and all the articles presented. The students had not previously experienced discussion as a method of learning mathematics, so one lesson was spent going through the process of how to discuss problems and use problem solving (Polya, 1957; A. Schoenfeld, 1985). Discussing problems is thought to support learning by helping students learn mathematical discourse practices (Stein et al., 2008). The class, divided into smaller groups, was presented with a problem which had to fulfil the following conditions. It had to be hard enough that the students did not arrive at the solution immediately; the explanation of the problem had to be simple to understand, but also contain subtle nuances that needed to be discussed; and it had to support a feeling of progression throughout. The problem chosen was as follows:

> You are standing in front of a 100-storey building. There is a staircase running all the way to the top, and at each floor you have access to an open window. You have two identical bowling balls that will both shatter when dropped from a given floor and all the floors above it. Your task is to find the lowest floor that will cause the bowling balls to break. The bowling balls can break at floor one, at floor one hundred, or any in between. Your method of finding the floor is evaluated according to the lowest number of tests you need to conduct in the worst-case scenario to find the lowest floor.
> Example:
>
> If you chose to drop the ball at every floor starting with the first floor, you will find the floor, but in the worst-case scenario you will have to test for all one hundred floors. If you drop the first ball at floor fifty and it breaks (worst case), you must then use the other bowling ball to test for the remaining 49 floors below starting at floor one. The worst case here is 50 tests (lowest floor

is either 49 or 50), already quite an improvement. How much better can you do?

The problem is easy to visualise, but the solution is not easy to unravel. From experience, the problem also quickly generates questions such as 'can I move up two or more flight of stairs between each drop?' and 'can I go down a floor if needed?'. Additionally, many students do not consider the worst-case scenario when presenting a solution, which, alongside these questions, illustrates the need to understand a problem before you can start to plan (Polya, 1957). The planning often started with trial and error, then possible answers being suggested to the group. The students had to argue for their solutions or ideas to the rest of the group so they would agree to present a possible solution to the teacher (Stein et al., 2008). The students argued for a decreasing number of tests as time went on. The progression towards the optimal solution to the problem could be seen, as the initial starting value of 100 tests quickly dropped to 50, 34, 27, or 24 and often ended at 19. The benefit is that there is very often a more efficient way to divide the problem, and the students started to see the method develop as they worked on it. The solution to this problem is in the appendix.

After this exercise, the researcher and class together set up the characteristics for a mathematical discussion on the board, including words such as suggestion, listening, verification, exploration, arguing, trial and error, and explanation. When the students were given the MPP, they were reminded of these traits to ensure a good discussion. The discussion between students followed the signs of a mathematical discussion to varying degrees. The periods of discussion that did not follow the signs were often of a non-mathematical nature.

Of a total of nine lessons in which MPPs were used, six were recoded and transcribed. The recording of the remaining three lessons failed due to technical difficulties. The six lessons that were recorded covered the following themes:

- Solving the quadratic equation
- Investigating probability (introductory lesson to loops)
- Plotting a graph (introductory lesson for plotting functions)
- Solving zero points of functions using the bisectional method
- Calculating and plotting the derivative of a function numerically
- Solving zero points of functions using Newton's method

The three lessons with technical difficulties were:

- Simulating a coin toss/dice throw through a Monte Carlo simulation

- Calculating and simulating Bayes theorem in probability
- Plotting the numerical solution to first order linear differential equations

Several types of data collection methods were used during PRIME. When implementing the pilot (autumn of 2017) video recordings, anonymous logs kept by the students, group interviews with the students, and researcher logs from the lessons were used. Analysis of this data provided the basis for the design of the second iteration. When the second iteration was implemented (spring of 2018), the data collection consisted of video and audio recordings of the students in the classroom as they worked on the task in addition to researcher logs from each lesson. The data were then analysed for the third revision (presented in Article 1). After the second redesign of the MPPs, the final round of implementation (autumn of 2018 until spring of 2020) was conducted.

The pilot and second implementation were recoded using physical cameras and microphones set up in the classroom, with selected cameras recording selected groups of students. After the lesson, the recordings were reviewed for quality. It became apparent that the students' screens were difficult to see properly. The reflection from various lights in the classroom, as well as the fact that the student's editor had a black background, made it almost impossible to investigate anything but the students' discussion. For the third iteration, a different approach was used. The difficulty of viewing the students' actions on screen was overcome by using screencast programs. The recoding was performed using the program FlashBack, which recorded the student's voice and computer screen, enabling the combination of voice and programming to be contextualised. The students were also in charge of the recording, so they could turn it off during breaks and if they were carrying out unrelated activity on their screen, such as checking Facebook or reading emails.

For each of the four classes, the students were asked to participate in the research. They were all given an outline of what the research entailed together with detailed information regarding the data collection. All the students recorded were informed and gave their written consent according to the Norwegian Centre for Research Data (NSD) rules for data collection, and PRIME has received approval from the NSD.

The aim was to record two to three groups from each class for data collection. This upper limit was chosen due to the technical limits of cameras, wires, microphones, and so on. The time it took to set up the equipment for each lesson was also a limitation, as there was only a 15-minute break between lessons. The criteria for the selection of the groups were as follows.

- All members of the group had to agree both to being recorded and to the use of their communications in the research project. Most, but not all, of the students in both classes agreed to being recorded. Since the entire group had to agree, quite a few groups were eliminated due to this criterion alone.
- The group needed to discuss their ideas through conversation throughout the lesson. If a group did not vocalise members' thinking, it was difficult to follow their train of thought and the recordings were discarded.

From those students remaining after the initial restrictions, two groups of two to four students were formed from each of the two years. The number within each group recorded varied due to absences.

4.4 Analysis procedure

Each member of the group recorded a data file with their voice and screen capture. All the students recorded wore a lavalier microphone that recorded their voice. Due to some technical difficulties, not all the recordings were usable, but since each student created their own files, no recorded lesson was omitted. The microphone also recorded the voices of the other students in the group, enabling each microphone to contribute to the transcription of the entire group. If one of the voices was unclear at any point, there were usually two to three other recordings to verify what was being said. To avoid having to look at four different data files for transcription, all the individual recordings from a group were assembled into one recording by using Camtasia. The voice from all four lavalier microphones were stacked, making each voice very clear, and each screen capture was as seen in figure 21.

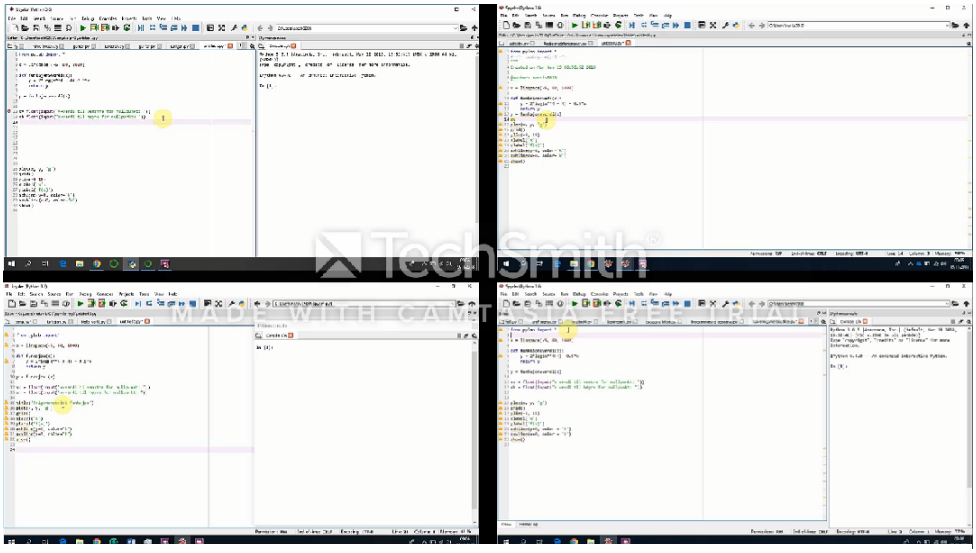


Figure 21: Result after combining four videos into one.

After the data files were combined into a single video file, all the audio from each group was transcribed in two steps. The first was an overview, in which the discussion within the group was dissected into smaller sections with a theme and a timestamp. An example is 'from 5:30 till 8:26, discussion regarding invalid solutions to quadratic equation'. Thereafter, the entire audio was transcribed with timestamps at regular intervals to ensure that the length of each part of the discussion was tracked. The importance of this is shown in Articles 2 and 3. The transcript was then sorted into the frameworks created for this work. The two frameworks in this thesis were created abductively by building on theory within the field of mathematical education as programming in the mathematics classroom is relatively new. There is one framework for the interactions between students when working on the MPP, as presented in Article 2, and another for the adversities encountered by the students when working with programming, as presented in Article 3.

The framework process started with open coding following axial coding (Strauss & Corbin, 1990). The advantages of this procedure are that it starts by being open-ended, allowing new concepts to reveal themselves. In Article 2, which focusses on exploratory talk and adversity, these two themes were categories within the framework before the open coding started. Through the coding, it became apparent that there were many sub-categories, and through an axial coding, both exploratory talk and adversities were divided into sub-categories, allowing new connections and new themes between concepts and categories to reveal themselves (Ryan & Bernard, 2003). This step was especially important in PRIME, as little previous research has been carried out on the topic. The framework for Article 2 is presented in table 1, where the two main categories are exploratory talk and adversity. Exploratory talk consists of several sub-categories, with exploration and explanation the two most common. Through the open coding, these were aligned with existing mathematical theory to build the abductive framework.

Table 1: The framework for article 2.

Code for interaction	Description
Exploratory talk	Engagement within the group consisting of ideas, suggestions, challenges, and justifications
Initiation	Start of a segment
Explanation	Explain and argue for solution
	Present criticism and suggestions
	Validate solution
Exploration	Testing code
	Running program
Agreement	Reaching common ground
Adversity	The group displays uncertainty over how to proceed
Positive	Leading to or facilitating exploratory talk (epistemological obstacles)
Negative	Leading to frustration and a 'this is not going to work' mentality (didactical and/or ontogenic obstacles)

In Article 3, the same procedure for building a framework was used. The focus was adversity, but here the open coding allowed for the different categories to reveal themselves. After the categories had been defined, axial coding was used to relate the different types of adversity. Article 3 investigates adversities encountered by students when working on programming in the mathematics classroom, where the framework is a combination of four elements. The four elements were built through open-coding and further specified through both axial coding and theory. The theory uses the barriers mentioned by Ko et al. (2004) and obstacles described by (Brousseau, 1997); see table 2.

Table 2: The framework for Article 3.

Type of adversity	Description
Concept	Unknown command Unable to recall command Unable to recall function of a command
Syntax	Placement of structure within a program Defining variables Structure of if statements Misunderstanding the sequential reading of code
Output	No output Understanding errors Not understanding errors Unexpected answer
Coding	Converting known mathematics into code Expanding program

For a more in-depth review of the two analytical frameworks, see Articles 2 and 3. This thesis draws on these frameworks in the discussion as they influence the recommendations for the design of MPPs in different ways.

Transcript analysis is an exploratory, qualitative methodology (Garrison et al., 2006). The two frameworks presented and their accompanying coding scheme (tables 1 and 2) were used in the analysis. The coding schemes were designed to be both meaningful and explicit in order to have reliability (Garrison et al., 2006). Meaningfulness comes from the open coding, as the coding scheme is a result of the observed interactions between students. Reliability comes from the structure and categories of the schemes, as each category was both broad enough to account for any interaction and narrow enough to be assigned to a given section of transcript. As coding is a time-consuming and challenging task, effort was made to keep the coding scheme parsimonious. As 'qualitative analysis needs to be well documented as a process' (Miles & Huberman, 1994, p. 12), the following section presents and exemplifies the process. Although Articles 2 and 3 use a different framework, they were both analysed in the same way and, using Article 3 as an example, the analysis was performed as follows.

Initially, the entire transcript was read with the overarching theme of the article in mind. As Article 3 investigated adversities, the entire transcript was read and each time a possible adversity presented itself, it was added to a list with a short description. At this time, the categories were not finalised so the descriptions were

very detailed. After the entire transcript of all the lessons was read, the descriptions were sorted into similar adversities, such as command adversities. Through this inductive process, and in combination with frameworks from the literature, a pilot version of the framework was created.

At this early stage, the framework contained many categories in an effort to ensure that every adversity was categorised. As the process continued, categories were revised several times, with revisions prioritising the simplification and reduction of categories. As an example, the command adversity category initially consisted of one category for forgetting a command and one for forgetting the function of a command. These were later merged to form one category.

With the revised framework, an analysis was performed, initially focusing on a quantitative distribution of the different categories, which allowed for a new lens in evaluating the framework. As an example, the first analysis in Article 3 contained a separate category for the editor of the program as it was both thought to contribute to adversity and had been observed. After the initial analysis, this was revealed to make a minimal contribution compared to the other categories, only appearing two times, neither of which was directly related to either the design of the task nor the orchestration of the lesson. To avoid diluting the adversity term over too many categories, editor adversity was removed from the framework. This is not to say that it should not be investigated but, rather, that it was not within the scope of this work.

After this second revision of the framework, the final version for the article was created. As this is a relatively new field within mathematics didactics, it is important to view the frameworks for what they are, namely a first attempt at investigating their respective fields. The analysis of both Articles 2 and 3 consisted of an initial frequency analysis followed by a qualitative analysis of the transcription; thus, they veer towards a mixed-method style of research. Considering the low number of participants, the thesis is primarily a qualitative study, with small elements of a quantitative study. The primary reason for the quantitative method is to illustrate the internal distribution of, for instance, adversity, and not the actual number of adversities observed. Both articles present the quantitative analysis first as doing so offers a good overview of the result before a more in-depth qualitative analysis is performed. The qualitative analysis of the two empirical articles uses segments of transcript frequently as this allows the reader an insight into the background. The qualitative analysis is detailed and refers to both the transcript and the observations made by the researcher to convey an accurate representation of the situation presented.

4.5 Ethical considerations, reliability, and validity

Several ethical issues are important to address in this work. The first is the duality of inhabiting the role of both teacher and researcher, which is both rewarding and challenging. The second refers to the more formal and well documented structures in qualitative studies which concern anonymity, influence on the subjects, data collection, and bias. Finally, the origin and financial support for PRIME is discussed.

As a teacher, I am performing insider research and have a preunderstanding of the school. Preunderstanding consists of knowledge, insights, and experience within the school (Brannick & Coghlan, 2007). This experience is not explicitly shown, but it permeates the research. Three examples are presented here to illustrate the advantages of performing inside research. First, knowledge of and access to the students allowed for more frequent changes to the task design. As the MPPs were introduced in the classroom, the students began to discuss programming outside the designated lessons. This allowed the teacher to participate in a discussion regarding the structure and difficulty of the MPPs outside the transcript, and several smaller changes to both the lessons and the MPPs themselves were made because of these discussions.

Second, knowledge of the individual students allowed for greater accuracy in the transcripts. With an in-depth knowledge of each student's mathematics proficiency and method of communication, transcription accuracy increased when the students referred to previous lessons or procedures, particularly when research addressed the adversities students encountered. Students have different ways of communicating frustration, and knowledge of their personality helped in the understanding of their recorded reactions.

Third, knowledge of the curriculum and the structure of all lessons taught throughout the year made the references the students used easy to understand. Additionally, MPPs were easier to implement since the structure of the entire year was up to the teacher, which allowed for the curriculum to be moved around to better fit with programming. All three types of knowledge, together with the fact that every aspect of the implementation was directly controlled and influenced by the researcher, made the research possible.

One of the main challenges is expressed perfectly in (Feldman, 1994), who presents a science teacher's dilemma regarding research in their own classroom, namely that they wish to collect precise and sufficient data to draw a conclusion, but the data collection is never precise nor sufficient to warrant validity. There will

always be too many variables unaccounted for that influence the educational situation. Some of the variables that could influence the research in PRIME are

- The teacher as the grade setter
- The teacher as responsible for the lesson and its orchestration
- The students displaying motivation due to recordings
- The researcher's bias in the data collection and analysis of the material
- The researcher's knowledge of the students affecting the interpretation of the transcript

Additionally, there was a plethora of small effects, such as classroom interaction, knowledge of the school, previous experience as a teacher, and so forth. To maintain the validity and accuracy of the research, the method and analysis are presented with an in-depth explanation of the procedures used. This should make the method and its implementation, and the analysis and frameworks used in the analytical process, clear and repeatable in other settings. Below, many of these variables are explicitly discussed.

According to the guidelines for research ethics in the social sciences, humanities, law, and theology, the individual has interests and integrity which cannot be set aside in research in order to achieve greater understanding or benefit society in other ways (NESH, 2016). This sentence should be interpreted as widely as possible. When introducing programming into the mathematics classroom, the pupils have a right to receive the same education as those who are not part of PRIME. According to the subject curriculum for science mathematics in Norway, the students have a right to receive 140 hours of mathematics (Udir, 2006). In the strictest sense, this prohibits the introduction of programming into the classroom, because it will reduce the number of lessons the students receive in mathematics[2]. To uphold their right in this regard, the students received additional lessons to maintain the correct number of hours for mathematics. This, however, created its own challenge. With an increased number of lessons, the students experienced an increased workload, which could have affected their schoolwork in other subjects. The design of PRIME, regarding the teaching of programming, needed to be organised in a way that minimised student workload. Designing PRIME so that the

[2] At the time of implementation for this thesis, the curriculum did not contain any requirement for programming.

students received ten extra lessons in programming during their normal school year allowed for a minimal influence on their schoolwork. Throughout the year, the students received programming tasks to both maintain and advance their skills. These tasks had to be voluntary and, at the same time, meaningful and beneficial, so that the students saw the value of completing them. The voluntary aspect is covered in the ethical guidelines, which stipulate that is has to be free, which means without external pressure or constraints on individual freedom (NESH, 2016). Many factors affected the students' motivation to undertake these tasks. Since the researcher was teaching the class and therefore responsible for the students' final grade, there was a danger that the students felt obligated to do these tasks in order not to receive an unfair evaluation. This eventuality is, again, covered by the ethical guidelines, which state that the researcher has a special responsibility for protecting the integrity of the individual (NESH, 2016). The neutrality of the researcher was maintained through the co-creation of tests and co-grading with fellow teachers at the school.

Data were collected through voice and screen recordings of the student's laptop screen when they were working on MPPs in lessons. When a student or group of students is recorded, their responses will be an entanglement of facts and values (Putnam, 1992). I will not go into the arguments related to such entanglement, but it is important to remember the effect this can have on the responses given. As the data collection was performed by a software program installed on the students' school laptops, the students could control when the recording started and stopped. The recorded students were all asked to initialise the recording at the start of the lesson but shown how to pause it if they wanted to access sensitive information on their laptop. The students often forgot that the recording was on, but thankfully no sensitive information was recorded.

Bias can be defined as a deviation of a measurement from the true value, and many research projects are in danger of bias (Kirch, 2008). Bias can originate from many different sources, but it is always a result of human choices. Interpretation bias refers to the data analysis being compromised so the result may not be valid. The researcher occupied two roles, teacher and researcher, possibly affecting the analysis of the data collected. To maintain the validity of the research, the frameworks used have been presented in detail, with several examples of how the coding was undertaken. When interpreting transcripts, in particular, excepts are presented to visualise the coding process in detail.

Selection bias refers to the selection of individuals for analysis being such that proper randomisation is not achieved and the population it is intended to analyse is therefore not represented. This work is affected by selection bias in two ways.

Firstly, since the number of participants is low, general conclusions are difficult to draw. In Articles 2 and 3, this is accounted for by explicitly pointing out that further research with a larger number of participants is required to confirm the findings. Secondly, the students do not reflect the average student in upper secondary school as they elected both science mathematics and to be a part of PRIME. The class, while having a good grade average, received every grade along the spectrum. Additionally, only two students had any prior experience with programming, and those students elected not to be a part of PRIME.

The detailed description of the method used in PRIME aims to increase both the validity and the reliability of the research performed. Validity is the degree to which the method measures what it purports to measure (Oluwatayo, 2012). Internal validity relates to causality, which concerns whether the result of the research performed is convincing (Bryman, 2016). The method chapter has described the process from the setting and selection of participants, data collection and transcription through analysis to ensure maximum transparency. One issue which might undermine internal validity is the availability of the data: not all data could be collected because doing so would violate the anonymity of the participants.

External validity concerns whether the results are generalisable beyond the context of PRIME. The important variable here is the type and number of students in PRIME. As noted, the students selected the class and therefore had an inherent motivation to commit to the MPPs given. I would argue that, with the iterative process of design and the fact that the students experienced both adversities and successes, the research is applicable to other mathematics classrooms.

Reliability is concerned with whether the same result would be obtained when the research was performed in a different setting (Bryman, 2016). Reliability is interesting, as there is a duality to consider. No two classrooms are alike, but there are nonetheless similarities among them. The classes in PRIME are not representative classes, as all the students elected science mathematics, indicating an interest in – or, at least, not a dislike of – mathematics. I would still argue that the results from PRIME are applicable to almost every mathematics classroom as students experience joy, frustration, and adversities and can participate in discussions with their peers in all classrooms. The results of PRIME are quite general, so the individual designer (or teacher) can apply the recommendations to their own classroom. Not every recommendation will be applicable to every class, but that was not the intention of the project.

The researcher in this PhD project is employed by the educational administration in Oslo (UDA), and UDA is sponsoring the entirety of the doctorate

degree. The funding received from UDA raises the ethical issue of the independence and transparency of this research. It is clear that UDA expects results to come from their funding, but it remains important that dependence does not undermine researchers' impartiality and the scientific quality of the research (NESH, 2016). UDA has not received any results prior to publishing, and there has been no pressure to present a given result, nor the expectation of one. This position is in agreement with the ethical guidelines, which state that 'the researcher is protected against undue pressure from the commissioner to draw particular conclusions, and is free to discuss alternative interpretations of their findings, or to point out scientific uncertainty' (NESH, 2016, p. 36). The only expectation on the part of UDA has been to receive an occasional presentation of progress and findings so far and the sharing of the accumulated knowledge with other teachers.

5 Summary of articles

The three articles form a network which feeds into the research question addressing what recommendations should guide the design of MPPs to facilitate mathematical learning in upper secondary school? from different angles. Each article builds towards a greater understanding of how a possible design for the implementation of programming in the mathematics classroom can be performed. Article 1 consists of the development of a structure for designing MPPs. Article 2 investigates the implementation of MPPs in the classroom, with a focus on what facilitates and hinders exploratory talk. From the work presented in Article 2, it became apparent that the adversities encountered by the students when working on MPPs needed to be investigated, and Article 3 presents this investigation. Both Articles 2 and 3 influenced the iterative development of the MPPs. Figure 22 illustrates the research process, whereby Article 1 investigated the design of the MPPs, Article 2 investigated the implementation of the MPP from Article 1 in a mathematics classroom, and Article 3 focused on one of the issues presented in Article 2, namely adversities occurring when working on MPPs. All the articles influenced the iterative process of developing recommendations for the implementation of programming into the mathematics classroom.

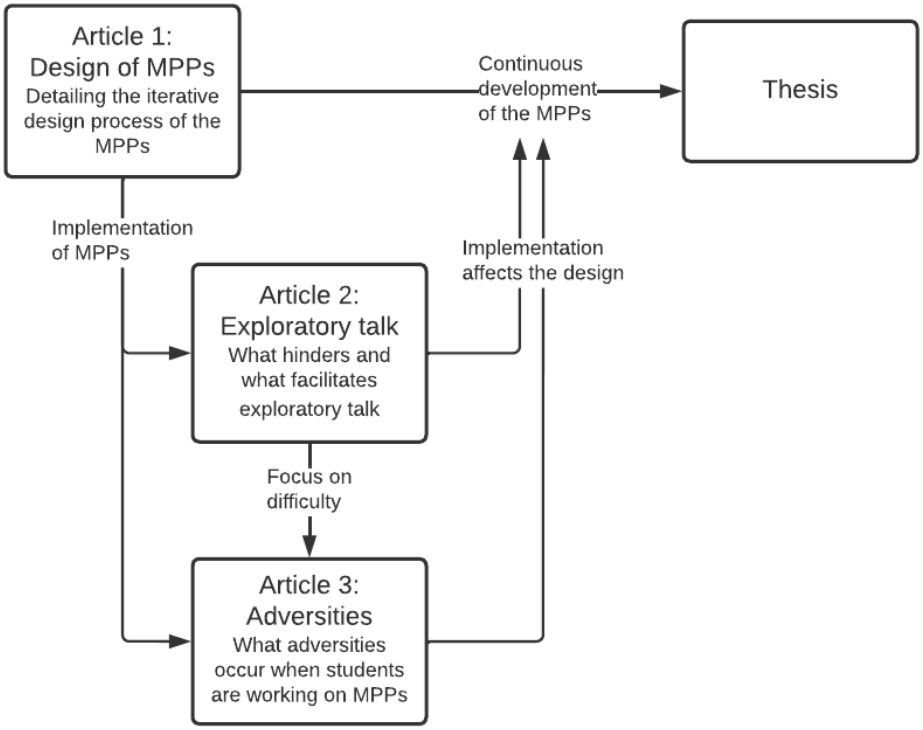


Figure 22: Illustration of how the three articles are connected and how they affect PRIME.

5.1 Summary of Article 1

The first article presents and discusses design ideas for problems implementing text-based programming as a tool for learning mathematics by incorporating both adidactical situations and problem-solving strategies. The design of MPPs uses the combination of problem-solving and adidactical situations as structures for design. The article presents an in-depth example to illustrate the seven-part design structure of the MPPs shown in figure 8. The first part enables the students to recall the previous mathematical knowledge required to engage with the problem, building the basis of the adidactical situation. The second part presents a limitation or challenge with a known mathematical concept. The article uses the vast number of mathematical methods of finding the zero point for different functions as an example and asks the students to reflect on the possibility of finding a single method for all zero points. The adidactical situation is achieved through a problem which, albeit time-consuming, the students can solve. The third part asks the

students to reflect and build a mathematical argument for the solution to the problem in part two.

Parts two and three build an adidactical situation, often with the inclusion of an epistemological obstacle, as students use their previous mathematical knowledge to build a new argument. Part four, like part one, enables the students to recall previous programming knowledge. This is the part of the MPP where the students start to program. Part five makes the students use the mathematical procedure or strategy they developed in parts two and three in a programming environment. This does not necessarily mean coding, but rather algorithmic thinking, such as breaking down the problem into steps the program can perform. The sixth part is the building of the program, using the mathematical strategy together with the algorithmic thinking from part five. This part is often time-consuming since the students are programming novices. At the end of part six, they have a working program that runs and yields an answer. The seventh part requires them to explore the limitations and affordances of the program through a mathematical lens. The article builds and argues for a structure for designing MPPs that facilitates adidactical situations in the mathematical classroom. The structure, together with the example, illustrates how mathematical learning can be blended with programming in upper secondary school mathematics.

5.2 Summary of Article 2

The second article investigates exploratory talk amongst students when working on MPPs. The article investigates elements contributing to and hindering exploratory talk. In six lessons, the audio and computer screens of several students were recorded as they worked on MPPs. The recordings were then transcribed and coded using the abductively created framework described in section 4.4. The first coding category was whether the communication was related to mathematics, programming, or a combination of the two. The second coding category was whether the transcript included adversity and/or exploratory talk. Both adversity and the separation between positive and negative adversities are presented as explained in section 3.3.

Exploratory talk, which can occur directly or because students encounter an adversity, is taken to lead to learning and presented in section 3.1. Together with several excerpts of transcriptions to illustrate the different category combinations, the results are presented as a chart for each lesson indicating the frequency of each category. In the lessons during which the mathematical theme to be programmed was familiar to the students, a significantly higher number of exploratory talks

versus adversity was observed. The exploratory talks taking place were also focusing on mathematics. In the lessons in which the mathematical theme to be programmed was unfamiliar (or new), adversity increased and more time was spent discussing programming code than mathematics.

The article discusses possible causes and implications of adversity and exploratory talk in the lessons and ends with three main findings. (1) Implementing programming in the mathematics classroom can facilitate mathematical exploratory talk, which facilitates learning. (2) Programming is best implemented to facilitate in-depth learning of already known mathematical concepts, as this initiates exploratory talk. More care is needed when utilising programming to teach new mathematical concepts, as this can result in an insurmountable adversity for the students. (3) Adversity is both important and challenging when programming is implemented in the mathematics classroom: important in that it can facilitate mathematical epistemological obstacles and challenging in that it adds another layer of complexity. This latter adversity is possible to overcome through task design and evaluation of the mathematical method to implement.

5.3 Summary of Article 3

Article 3 investigates which types of adversities are encountered by upper secondary school students when working with MPPs in the mathematics classroom and how the adversities are related to the learning of mathematics. The transcripts from lessons were reviewed and divided into segments where the students encountered an adversity which they either solved or failed to solve. Each segment was then coded according to an abductively created framework (see section 4.4) consisting of four categories:

1) Concept adversity is any obstacle related to the use and knowledge of different commands and types in the programming language.
2) Syntax adversity includes the structure of conditions and loops and the logical build of a program.
3) Output adversity occurs when pressing 'run the program' button presents an obstacle. This can take many forms, from syntax errors and unexpected answers to no output at all.
4) Coding adversity occurs when the students are attempting to convert a mathematical procedure to programming code.

The framework is built upon a combination of known and observed programming barriers and mathematical obstacles, as explained in section 3.3.

Concept adversities are frequent, but since each incident is quickly resolved through either discussion within each student group or through a short question to the educator, they do not affect the students' work in a significantly negative way. Such adversities can be reduced by giving the students skeleton code or ensuring a repetition of previously used code, but the research also indicated that this problem diminishes as the students become more comfortable and experienced with the programming language. A possible alleviation of this type of adversity is to give the students a glossary of codes, with a short explanation of each command. These adversities were not viewed as a major problem in the lessons conducted.

Sequencing adversities were the least frequently seen type and, when resolved, were observed to facilitate an internal construction of the programming code and understanding of each element of it. Additionally, students were observed to discuss and argue for the logical procedure of the program. In one of the MPPs, where the students were familiar with the mathematics before applying programming, resolved sequencing adversities were prominent. In the second MPP, where the students were required to apply a previously unknown mathematical method and programming, they were not able to resolve the adversity. This outcome indicated that when implementing programming in the mathematics classroom, it is better to apply programming as a tool for deep learning when the mathematics has already been taught, rather than as a means to introduce a new topic.

Output adversities were as prominent as command adversities but initiated a discussion over a longer time span and were observed to contain more mathematics, often combined with programming. When output adversities were not resolved within a reasonable timeframe, the students gave up and deleted the code causing the adversity. This made them disengage from the adidactical situation (Brousseau, 1997). Output adversities are difficult to mitigate since they are hard to predict, and students have limited knowledge of how to interpret feedback from the editor; however, teaching the students basic knowledge of how to handle errors was particularly advantageous.

Translation adversities were, for the mathematically inclined, perhaps the most interesting, as they make visible the combination and advantage of mathematics and programming. The two main features of translation were to (1) use mathematics to make the program understand a simple mathematical relationship; and (2) use mathematics to solve a complex problem through building a short program. Preventing translation from becoming an ontogenic obstacle depended on the students' building of a mathematical model into a programming model in which the elements and their relation were essential for the solution (Ko et

al., 2004). Facilitating this building, the task needs to scaffold the intended problem by using and recalling the previous mathematical knowledge required and guiding the students in their exploration to assemble a new mathematical and/or programming model (Kirschner et al., 2006; Reiser, 2004).

6 Discussion

The following chapters use the articles and the connections between them to answer the research question concerning which recommendations should guide the design of MPPs to facilitate mathematical learning in upper secondary school.

The aim of this thesis is to present a set of recommendations for the design of mathematical tasks in which programming is an integrated part. Several factors influencing these recommendations are derived from both previous research and the research undertaken for this thesis. Last time programming was implemented into the school setting, it did not succeed, partly due to the students not being able to connect the programming they performed with the intended mathematical ideas (Misfeldt & Ejsing-Duun, 2015; Papert, 1980). To prevent a similar result this time, the connection between the programming the students undertake needed to be explicitly linked to mathematics. PRIME has, since its conception, been explicit that programming is a tool for learning mathematics and that it is only used when it facilitates mathematical learning. When building a program plotting, solving, and calculating properties of quadratic equations (Articles 1, 2, and 3), the building facilitates the students' learning

- challenges with negative roots and invalid solutions
- the link between the graphical representation and the above challenges
- the generalisation of finding properties of quadratic equations, such as extreme values, the derivative, and zero points.

Additionally, the logical and structural build of a program is closely related to the structure of mathematics. I argue that the students, by applying a program directly to a mathematical concept, experienced a direct link between the programming and mathematics (Article 2). In PRIME, this link was accomplished through the design of the task. As previously stated, tasks are the main 'thing to do' in the mathematics classroom (Watson et al., 2015, p. 3), and task design is therefore central in facilitating the implementation of programming into that classroom. For a task to be successful in PRIME, I defined several criteria from the work in the three articles, which consider the structure, discussion among students, and adversities encountered by students, respectively. The combination culminates in what distribution of these criteria is beneficial to learning mathematics. From the

work conducted in PRIME, a set of recommendations was developed, each of which comes with its own argument.

Design tasks where programming is a tool for learning mathematics.

When programming is implemented within mathematics rather than taught as a separate subject, the focus should remain on the students' learning of mathematics. It is easy to design a task in which the students use programming to build a small game or give instructions to a movable object, such as a car, with the connection to mathematics being obscure. In fact, programming is excellent at engaging the students in such tasks (Forsström & Kaufmann, 2018; Kjällander et al., 2021). The challenge becomes how to facilitate the link between programming and mathematics to avoid a similar fate to that seen last time programming was implemented (Misfeldt & Ejsing-Duun, 2015).

For programming to be a valuable artifact for both students and teachers, its use must be clear and obvious. The MPPs presented in PRIME exemplify this need for clarity by making the mathematical challenge clear very early in the task (Article 1), seasoning the code building with mathematical tasks (Article 1), facilitating the students' mathematical exploratory talk (Articles 1 and 2), and using the program to explore limitations and evaluating mathematically why such limitations occur (Articles 1 and 2).

Articles 2 and 3 indicate that there needs to be a concrete and obvious link between the activities in which students are engaging and the subject (in this case, mathematics). Building a program which requires mathematics is likely to build and consolidate the link between programming and mathematics. The connection between mathematics and programming does not always have to be explicit; however, in upper secondary school in particular, it is important that the task design always includes the link to mathematics when programming is engaged in. The MPPs should, as presented in Articles 1, 2, and 3, focus on learning and exploration (Knight et al., 2017), which gives rise to the next recommendation.

Ensure MPPs facilitate exploratory talk.

As mentioned in Article 2, multiple studies indicate that facilitating mathematical discussion in a classroom is closely linked to mathematical learning (e.g. Cobb et al., 1997; Kazemi & Stipek, 2009; Nathan & Knuth, 2003; Resnick et al., 2017; Sfard, 2000). Article 2 not only presents the challenges that programming brings to students' work, but indicates that students participate in exploratory talk when engaging with MPPs. When exploring together, the students have a larger

network of abilities from which to draw knowledge when faced with an obstacle, as seen in Articles 2 and 3. In this situation, it is hoped that they will build an epistemological obstacle, and the number of ontogenic and didactical obstacles encountered will be limited (Brousseau, 1997). When they engage in exploratory talk, the students show greater stamina (Articles 2 and 3), are more prone to build on each other's ideas and suggestions (Article 2), and present adversities to the group for help or discussion before seeking outside help (Articles 2 and 3). When the MPPs contain tasks facilitating discussions, students are more likely to effectively engage with their peers (Darabi et al., 2013). The tasks that explicitly stated 'discuss with your neighbours' were also designed to incorporate problem solving (Polya, 1957; A. Schoenfeld, 1985), as discussed in section 3.3.

Closely linked to problem solving is algorithmic thinking, in which students decompose a problem into smaller tasks, solve the tasks individually, and finally assemble the parts to solve the problem (Abramovich, 2015; Lockwood et al., 2016; Stephens, 2018; Stephens & Kadijevich, 2020). Engagement with peers was recognised as the iterations of the MPPs developed: the first version contained little to no discussion about tasks or problem solving, whereas the version presented in Article 1 included several tasks in which both discussion and problem-solving were present.

It is advantageous to engage in programming after having learnt a mathematical theme, rather than using programming to learn a new mathematical theme.

Article 3 addresses the double barrier students can experience if they are tasked with building a program using an unknown mathematical method, or, in other words, they are asked to learn a new mathematical theme using programming. As discussed in Article 3, two similar MPPs were presented, one of which relied on known mathematical concepts such as zero points, function value, and halving an interval (bisectional method). While the bisectional element of the MPP did create its own adversities, it did not make the students deviate from discussion or from working to understand the mathematical method the task was designed for them to uncover.

The second MPP used many of the same concepts but also included a formula in which the derivative of a function was used. This increase in complexity made the students deviate from the mathematics and focus on the programming, essentially relying on copying and pasting code segments. As complexity has been argued to be additive (Sweller, 2006), designing an MPP consisting of two adversities was not

successful. Rather than learning mathematics, the students engaged with the MPP to instrumentally build the code, and without understanding the underlying mathematics, they were unable to critically evaluate the output from the program. This argument gives rise to the following recommendation.

MPPs should mitigate non-mathematical adversities.

The design of the MPPs should implement adversities which students are able to overcome. As discussed in the previous paragraph, students encounter several different types of adversities when working with programming (see Article 3). The distinction between a mathematical and a non-mathematical adversity can be challenging as, for instance, adversities relating to the output of a program can be either. If the output is a syntax error, the adversity can be non-mathematical if caused by a wrong indent or mathematical if a calculation could not be performed due to a missing or misplaced parenthesis, as described in more detail in Article 3.

While it is helpful and advantageous to remove many adversities, it would be disadvantageous or impossible to remove all. As adversity, and particularly its resolution, is beneficial to learning (Hiebert & Grouws, 2007), I argue that one should strive to design adversities which students are able to overcome, thereby facilitating the adversity becoming an epistemological obstacle (Brousseau, 1997). Command adversities, for instance, are difficult to remove, but since they are resolved quickly (Article 3), they are not a major concern. It is more important to avoid designing problems in which the combination of code and mathematics becomes too complex.

MPPs should have a low floor and a high ceiling.

A 'low floor' is now a commonly used term to identify whether a programming language is easy to learn and the time taken from learning a few commands to building programs is short. In PRIME, low floor, in the context of task design, indicates whether a student will be able to initialise their work on the task (Papert, 1980; Sullivan et al., 2012). I argue that programming is the most complex digital tool introduced into the upper secondary school mathematics classroom, and to reduce students' (and teachers') frustration, it is essential that tasks have a low floor.

In all the MPPs, the initial tasks utilised the students' previous knowledge of mathematics as a gateway to the subsequent tasks (Articles 1, 2, and 3). The idea of a low floor is conceptualised as being that every student should be able to build an initial coding sequence that runs without encountering errors. This is,

unfortunately, almost impossible, since inevitably some students accidently enter a key into a code that should not be there, typically resulting in a syntax error. From Article 3, I argue that this type or error is quickly resolved, although it does often require assistance from the teacher. If the number of errors remains low, students will continue working on the MPP and not give up.

A 'high ceiling' is a commonly used term to describe a programming language that has a wide range of possible uses. In PRIME, high ceiling, in the context of task design, indicates whether a student has the possibility to expand the program throughout the problem (Papert, 1980; Sullivan et al., 2012). In Article 1, the high ceiling was only implemented towards the end of the MPP, but the analysis performed throughout Articles 2 and 3 showed the students discussing their own mathematical ideas and inquiring how to alter the program to facilitate these ideas. For instance, when working on the bisectional MPP in which the program found one zero point for each run, the students asked how to change it to locate every zero point for each run. A high ceiling is very similar to an open task, which has been shown to promote students' mathematical creative thinking and facilitate their realisation of their own intentions (Sullivan et al., 2015). Whether students experience a task as open or with a high ceiling has been argued to be a consequence of their previous educational history, the expectations of the teacher, and students' understanding of these (Brousseau, 1986). I argue that tasks with a high ceiling provide two additional benefits. When MPPs are implemented in a classroom, a high ceiling will allow high-achieving students to expand their program further with more complex procedures. For instance, when working on the MPP solving the quadratic equation, the last task encouraged them to expand the program to include the derivative and extremal values, to name a few. This possibility allowed them to use their mathematical knowledge and combine it with programming to expand the program.

The second benefit is that a high ceiling allows the students to gain ownership of their work, which has been shown to play a part in learning (Carpenter & Lehrer, 1999; Conley & French, 2014). In all the articles, ownership continues to be apparent, with the students altering the program according to their own preference. A very endearing, but still important, occurrence was the students' choice to colour their graphs when plotting them, which continued to elicit a positive reaction from the students (see Articles 1 and 2).

As every teacher knows, it is rare that a classroom activity unfolds in the same way in two classrooms. The same can be said of MPPs, since they are quite extensive

and complex in nature. I argue that a simple problem statement, such as solving an equation or plotting a graph, will, in most classrooms, be undertaken by students in similar ways. When the complexity increases, either because the problem requires several steps or one of the steps is difficult, there is a higher chance of dissimilar performance. The recommendations above attempt to combine mathematics and programming, both of which are subjects that many students find difficult (Campbell & Epp, 2005; Grover & Pea, 2013; Jenkins, 2002; Ko et al., 2004; Merenluoto & Lehtinen, 2004; Nelson & Powell, 2018; Piteira & Costa, 2013). The recommendations do not resolve all challenges related to the implementation of programming in the upper secondary school mathematics classroom but aim to be a starting point from which future research can proceed. They are intended as a step on the way to creating a set of design principles for task design when implementing programming in mathematics.

7 Concluding thoughts and future work

7.1 A happy marriage between mathematics and programming?

Through the work undertaken in this thesis, I hope to have shown that programming can be utilised to teach mathematics in an upper secondary school setting. This does not mean that there are not challenges, but rather that the potential for using programming as a tool for learning mathematics and the increase in possibilities it offers outweigh the adversity it brings. As we are currently in the infancy of the second round of introducing programming into schools, it is vital to bear in mind the need to progress slowly. Currently, students have little to no knowledge of programming when they enter upper secondary school, and it takes a significant amount of time to both teach them programming and use programming in lessons. As the years go by and students arrive at upper secondary school with more and more knowledge of programming, it will become simpler to arrive at the point where programming can be used in lessons rather than taught. In the years before that point is reached, however, it is important to limit the complexity of MPPs, as argued for in the articles. In the coming years, wider-scale research involving a much larger number of students should be undertaken to investigate the different aspects of the implementation of programming.

When combining elements from the presented task design, exploratory talk, adidactical situations, and minimising adversities through the same elements, programming can be utilised to facilitate mathematical learning in the classroom. The caveat is that there are many hurdles for the teacher to overcome to achieve this outcome. Task design is very time-consuming and presents many difficulties, such as what mathematical concept to apply programming to, how to utilise programming efficiently, how to facilitate mathematical learning, and, finally, how to avoid unnecessary complexity. Exploratory talk needs to be explicitly facilitated as few students can participate in a mathematical discussion without any form of guidance or training. Adversities influence and set the stage for both task design and exploratory talk in that the students through their work with the MPPs and the didactical contract with their teacher should overcome these adversities. The

restructuring of known concepts and introduction of new ones contribute to an increased network of mathematical knowledge and promote mathematical learning. By combining MPPs and classroom orchestration, it is possible to design lessons that facilitate mathematical learning using programming as a tool.

Although the work presented is well founded within research into mathematic didactics, I have no doubt that the recommendations in PRIME can be applied when implementing programming in other sciences.

7.2 Limitations

The students involved in this research all received a ten-hour crash course in programming at the start of the year to ensure that they had a basic knowledge of programming before starting work on the MPPs. While the MPPs also facilitated the students' learning of programming code, I argue that the initial knowledge they received enabled the students to shift their focus towards mathematics rather than allowing programming to become a dominating challenge. With the implementation of programming in the mathematics classroom, there is a need to teach the students programming in addition to mathematics.

While this work has only looked at how to implement programming in mathematics, it simultaneously strongly suggests that the implementation of programming will negatively affect the amount of time the students spend on learning mathematics. As argued previously, programming can, in several instances, contribute to the learning of mathematics, but there are two important caveats to this statement. Firstly, the students in this research all chose to participate, and it is thus not an unreasonable assumption that they were, at least to some extent, motivated to learn programming. Secondly, the students received an additional ten hours' tuition in basic programming. Even with these two advantages, the students still encountered challenges with programming, and there is a real concern that implementing programming in mathematics will take away precious time. A solution could be either to increase the number of lessons in mathematics to allow for the teaching of programming, or, and perhaps in preference, to implement a new course that focuses only on programming and related subjects, such as technology and logical thinking, as suggested by several other reports (Ludvigsen, 2015; Sevik, 2016). T

7.3 The entire school setting and future work

What about the rest of the school system? How can programming contribute to learning mathematics from primary school onwards? In this section, I present a sequence of examples to show how programming can contribute to learning mathematics and be a bridge between the mathematics curriculum in primary school, secondary school, and at university level. These examples are based on experiences of PRIME and my own experiences as a teacher.

In primary school, students learn simple programming routines using a simple language. They can, for instance, program a moving object (a car, a ball, or similar) to drive from one point in the classroom to another. Let us say, for the sake of simplicity, that this program only accepts inputs for movement in the form of forward, backward, turn left (90 degrees), and turn right (90 degrees). To drive, a sequence of code can be:

```
forward(10)
turn right
forward(10)
```

As more code sequences are used, the relationship between the code and the movement becomes increasingly apparent for the student, for example that the car drives ten steps forward and ten steps to the right. By applying similar code sequences, it is possible to reach any positive point.

The next step could be to mark the floor of the classroom with two tapes indicating forward and right, with steps along each tape. The tapes meet at the point where the car starts, namely the origin with coordinates (0,0). One could then say that the origin of the car is named with the coordinates (0,0), and the endpoint of the drive is named with the coordinates (steps forward, steps to the right), introducing the concept of coordinates. The next step is to program the car to travel between several points, for instance from origin to (10, 10) to (15, 5). This task initiates simple calculations to correctly program the route of the car. Depending on the grade of the students, the points can be only additive or include subtraction.

The students can also be introduced to the concepts of coordinate system, axis, and so forth. Furthermore, one could give the students the task of investigating how to move the car twice as far or half as far, thereby introducing the elements of multiplication and division. As the students program the movable object, initially along points along a straight line, they could be introduced to the concept of linear functions. A later variant could allow the students to program a course that traverses a set of points, but in a more fluent way, following a curved path rather than moving from point to point, foreshadowing the use of non-linear functions.

When the students later learn about coordinate systems, they can refer to the exercise they performed in the lower grades and make the connection of forward/backwards to positive/negative y-axis and right/left to positive/negative x-axis. The curved route is a polynomial, rational, or exponential function.

The addition of travel from point to point is an introduction to vectors, where the students can visualise that adding the distance from the origin to (10, 10) and from (10, 10) to (15, 5) is the same as travelling to (15, 5) directly, foretelling the addition of vectors, which are taught at secondary school. This is one small example of possible benefits of using programming throughout the school spectrum – to be able to foretell future mathematical concepts very early. There are many similar examples, and more will be uncovered as programming becomes more prominent throughout the school system. This, in my opinion, would be a truly valuable use of programming, where one strives to design lessons that both facilitate mathematical learning and foreshadow future mathematical concepts.

For those countries already implementing programming in the mathematics classroom, there is a need for research into how this effects and affects learning. The work carried out here is small-scale, and while it provides several interesting observations, several large-scale investigations need to be conducted. Possible research questions include:

- How has the implementation of programming in the mathematics classroom changed the orchestration of lessons?
- How does the use of programming affect the students' learning of mathematics on a large scale?
- How does programming affect the students' difficulties in mathematics?
- How does programming affect the students' motivation in mathematics?

Many other research questions will undoubtedly emerge. I am curious to see what future research into the implementation of programming in mathematics (and physics, biology, chemistry, and earth sciences) will bring.

8 References

Abramovich, S. (2015). Mathematical problem posing as a link between algorithmic thinking and conceptual knowledge. *Teaching of Mathematics*, *18*(2), 45-60.

Ahmed, G., Nouri, J., Zhang, L., & Norén, E. (2020). Didactic Methods of Integrating Programming in Mathematics in Primary School: Findings from a Swedish National Project. Proceedings of the 51st ACM Technical Symposium on Computer Science Education,

Andersen, L., & Cross, T. L. (2014). Are students with high ability in math more motivated in math and science than other students? *Roeper Review*, *36*(4), 221-234.

Anderson, T., & Shattuck, J. (2012). Design-based research: A decade of progress in education research? *Educational researcher*, *41*(1), 16–25.

Anthony, G., & Walshaw, M. (2009). Characteristics of effective teaching of mathematics: A view from the West. *Journal of Mathematics Education*, *2*(2), 147-164.

Artigue, M. (2015). Perspectives on design research: The case of didactical engineering. In A. Bikner-Ahsbahs, C. Knipping, & N. Presmeg (Eds.), *Approaches to qualitative research in mathematics education* (pp. 467–496). Springer.

Artigue, M., Haspekian, M., & Corblin-Lenfant, A. (2014). Introduction to the theory of didactical situations (TDS). In A. Bikner-Ahsbahs & S. Prediger (Eds.), *Networking of theories as a research practice in mathematics education* (pp. 47–65). Springer.

Avigad, J., Dean, E., & Mumma, J. (2009). A formal system for Euclid's Elements. *The Review of Symbolic Logic*, *2*(4), 700-768.

Balacheff, N. (1990). Towards a problématique for research on mathematics teaching. *Journal for Research in Mathematics Education*, *21*(4), 258–272.

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *The journal of the learning sciences*, *13*(1), 1–14.

Barab, S. A., Hay, K. E., Barnett, M., & Keating, T. (2000). Virtual solar system project: Building understanding through model building. *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching*, *37*(7), 719-756.

Barreto, H. (2015). Why Excel? *The Journal of Economic Education*, *46*(3), 300-309.

Berisha, V., & Bytyqi, R. (2020). Types of mathematical tasks used in secondary classroom instruction. *International Journal of Evaluation and Research in Education*, *9*(3), 751–758.

Berry, D. M. (2013). The essential similarity and differences between mathematical modeling and programming. *Science of Computer Programming*, *78*(9), 1208-1211.

Boaler, J. (1999). Participation, knowledge and beliefs: A community perspective on mathematics learning. *Educational Studies in Mathematics*, *40*(3), 259-281.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). The Nordic approach to introducing Computational Thinking and programming in compulsory education. *Report prepared for the Nordic@ BETT2018 Steering Group*, 397-400.

Bokhove, C., & Drijvers, P. (2010). Digital tools for algebra education: Criteria and evaluation. *International Journal of Computers for Mathematical Learning*, *15*(1), 45–62.

Bosse, Y., & Gerosa, M. A. (2017). Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning Mid-Stage. *ACM SIGSOFT Software Engineering Notes*, *41*(6), 1-6.

Brannick, T., & Coghlan, D. (2007). In defense of being "native": The case for insider academic research. *Organizational research methods*, *10*(1), 59-74.

Bray, A., & Tangney, B. (2017). Technology usage in mathematics education research–A systematic review of recent trends. *Computers & Education*, *114*, 255–273.

Brousseau, G. (1986). Fondements et méthodes de la didactique des mathématiques. *Recherches en didactique des mathématiques (Revue)*, *7*(2), 33-115.

Brousseau, G. (1997). *Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990* (Cooper M., Balacheff N., Sutherland R., & Warfield V., Trans.). Kluwer Academic Publishers.

Brousseau, G. (2008). Research in mathematics education. In M. Niss (Ed.). *Proceedings of the 10th international congress on mathematical education*, 244–254.

Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The journal of the learning sciences*, *2*(2), 141–178.

Brydon-Miller, M., Greenwood, D., & Maguire, P. (2003). Why action research? *Action Research*, *1*(1), 9-28.

Bryman, A. (2016). *Social research methods*. Oxford university press.

Buchberger, B. (1990). Should students learn integration rules? *ACM Sigsam Bulletin*, *24*(1), 10–17.

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of educational research*, *87*(4), 834-860.

Campbell, J. I. D., & Epp, L. J. (2005). Architectures for arithmetic. In J. I. D. Campbell (Ed.), *Handbook of mathematical cognition* (pp. 347-360). Psychology Press.

Carpenter, T. P., & Lehrer, R. (1999). Teaching and learning mathematics with understanding. In E. Fennema & T. R. Romberg (Eds.), *Mathematics classrooms that promote understanding*. Erlbaum.

Choi, J., & Walters, A. (2018). Exploring the Impact of Small-Group Synchronous Discourse Sessions in Online Math Learning. *Online Learning*, *22*(4), 47-64.

Cobb, P., Boufi, A., McClain, K., & Whitenack, J. (1997). Reflective discourse and collective reflection. *Journal for Research in Mathematics Education*, *28*(3), 258-277.

Cole, R., Purao, S., Rossi, M., & Sein, M. (2005). Being proactive: where action research meets design research. *ICIS 2005 Proceedings*, 27.

Conley, D. T., & French, E. M. (2014). Student ownership of learning as a key component of college readiness. *American Behavioral Scientist*, *58*(8), 1018-1034.

Corey, S. M. (1954). Action research in education. *The journal of educational research*, *47*(5), 375–380.

Darabi, A., Liang, X., Suryavanshi, R., & Yurekli, H. (2013). Effectiveness of online discussion strategies: A meta-analysis. *American journal of distance education*, *27*(4), 228-241.

De Lange, J. (1987). *Mathematics, insight and meaning: Teaching, learning and testing of mathematics for the life and social sciences*. OW & OC Utrecht.

Diković, L. (2009). Applications GeoGebra into teaching some topics of mathematics at the college level. *Computer Science and Information Systems*, *6*(2), 191-203.

Dreyfus, T. (2020). Abstraction in mathematics education. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 13–16). Springer.

Drijvers, P., Ball, L., Barzel, B., Kathleen Heid, M., Cao, Y., & Maschietto, M. (2016). *Uses of technology in lower secondary mathematics education: A concise topical survey*. Springer Nature.

Fagan, B., & Payne, B. (2017). Learning to Program in Python–by Teaching It! Proceedings of the Interdisciplinary STEM Teaching and Learning Conference,

Fauconnier, G., & Turner, M. (2008). *The way we think: Conceptual blending and the mind's hidden complexities*. Basic Books.

Feldman, A. (1994). Erzberger's dilemma: Validity in action research and science teachers' need to know. *Science education*, *78*(1), 83-101.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., & Solomon, CFeurzeig, Wallace. (1969). *Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project*. Cambridge, MA: BBN

Forsström, S. E., & Kaufmann, O. T. (2018). A literature review exploring the use of programming in mathematics education. *International Journal of Learning, Teaching and Educational Reseach*, *17*(12), 18–32.

Francisco, J. M., & Maher, C. A. (2005). Conditions for promoting reasoning in problem solving: Insights from a longitudinal study. *The Journal of Mathematical Behavior*, *24*(3-4), 361-372.

Garrison, D. R., Cleveland-Innes, M., Koole, M., & Kappelman, J. (2006). Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education*, *9*(1), 1-8.

Gerbic, P., & Stacey, E. (2005). A purposive approach to content analysis: Designing analytical frameworks. *The Internet and Higher Education*, *8*(1), 45-59.

Godino, J. D., Batanero, C., Contreras, A., Estepa, A., Lacasta, E., & Wilhelmi, M. (2013). Didactic engineering as design-based research in mathematics education. In B. Ubuz, Ç. Haser, & M. A. Mariotti (Eds.), *Proceedings of the Eight Congress of the European Society for Research in Mathematics Education* (pp. 2810-2819). Middle East Technical University.

Gravemeijer, K., Stephan, M., Julie, C., Lin, F.-L., & Ohtani, M. (2017). What Mathematics Education May Prepare Students for the Society of the Future? *International Journal of Science and Mathematics Education*, *15*(1), 105–123.

Gray, E. M., & Tall, D. O. (1994). Duality, ambiguity, and flexibility: A "proceptual" view of simple arithmetic. *Journal for Research in Mathematics Education*, *26*(2), 115–141.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43.

Han, B., Bae, Y., & Park, J. (2016). The Effect of Mathematics Achievement Variables on Scratch Programming Activities of Elementary School Students. *International Journal of Software Engineering and Its Applications*, *10*(12), 21–30.

Harel, G., & Sowder, L. (2005). Advanced mathematical-thinking at any age: Its nature and its development. *Mathematical thinking and learning*, *7*(1), 27–50.

Hiebert, J. (2013). *Conceptual and procedural knowledge: The case of mathematics*. Routledge.

Hiebert, J., & Grouws, D. A. (2007). The effects of classroom mathematics teaching on students' learning. In J. F. K. Lester (Ed.), *Second handbook of research on mathematics teaching and learning* (Vol. 1, pp. 371-404). Information Age Publishing.

Hillel, J. (1992). The computer as a problem-solving tool: It gets a job done, but is it always appropriate? In J. P. Ponte, J. F. Matos, J. M. Matos, & D. Fernandes (Eds.), *Mathematical problem solving and new information technologies* (pp. 205–218). Springer.

Hoffman, B., & Spatariu, A. (2008). The influence of self-efficacy and metacognitive prompting on math problem-solving efficiency. *Contemporary Educational Psychology*, *33*(4), 875–893.

Jenkins, T. (2002). On the difficulty of learning to program. Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences,

Johnson, H. L., Coles, A., & Clarke, D. (2017). Mathematical tasks and the student: Navigating 'tensions of intentions' between designers, teachers, and students. *Zdm*, *49*(6), 813–822.

Joubert, M. V. (2007). *Classroom mathematical learning with computers: The mediational effects of the computer, the teacher and the task.* [Doctoral thesis, University of Bristol].

Kazemi, E., & Stipek, D. (2009). Promoting conceptual thinking in four upper-elementary mathematics classrooms. *Journal of education*, *189*(1-2), 123-137.

Kieran, C. (2019). Task design frameworks in mathematics education research: An example of a domain-specific frame for algebra learning with technological tools. In G. Kaiser & N. Presmeg (Eds.), *Compendium for Early Career Researchers in Mathematics Education* (pp. 265-287). Springer.

Kirch, W. (2008). *Encyclopedia of Public Health: Volume 1: A-H Volume 2: I-Z*. Springer Science & Business Media.

Kirschner, P., Sweller, J., & Clark, R. E. (2006). Why unguided learning does not work: An analysis of the failure of discovery learning, problem-based learning, experiential learning and inquiry-based learning. *Educational Psychologist*, *41*(2), 75-86.

Kjällander, S., Mannila, L., Åkerfeldt, A., & Heintz, F. (2021). Elementary students' first approach to computational thinking and programming. *Education Sciences*, *11*(2), 80.

Knight, S., Rienties, B., Littleton, K., Tempelaar, D., Mitsui, M., & Shah, C. (2017). The orchestration of a collaborative information seeking learning task. *Information Retrieval Journal*, *20*(5), 480-505.

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, *92*(3), 170-181.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, 199-206.

Laborde, C., & Sträßer, R. (2010). Place and use of new technology in the teaching of mathematics: ICMI activities in the past 25 years. *Zdm*, *42*(1), 121–133.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *Acm sigcse bulletin*, *37*(3), 14-18.

Lampert, M., Rittenhouse, P., & Crumbaugh, C. (1996). Agreeing to disagree: Developing sociable mathematical discourse. In D. R. Olson & N. Torrance (Eds.), *The handbook of education and human development: New models of learning, teaching, and schooling* (pp. 731-764). Blackwell.

Lee, Y.-J. (2011). Scratch: Multimedia programming environment for young gifted learners. *Gifted Child Today*, *34*(2), 26–31.

Lehrer, R., & Schauble, L. (2005). Developing Modeling and Argument in the Elementary Grades. In T. Romberg, T. Carpenter, & F. Dremock (Eds.), *Understanding mathematics and science matters* (pp. 29-53). Erlbaum.

Lehrer, R. S., L. (2001). *Accounting for contingency in design experiments* Annual Meeting of the American Educational Research Association, Seattle, WA.

Lehtinen, E., Merenluoto, K., & Kasanen, E. (1997). Conceptual change in mathematics: From rational to (un) real numbers. *European Journal of Psychology of Education*, *12*(2), 131.

Leung, A., & Baccaglini-Frank, A. (2016). *Digital Technologies in Designing Mathematics Education Tasks: Potential and Pitfalls* (Vol. 8). Springer.

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: logo vs. scratch. *Proceedings of the 41st ACM technical symposium on computer science education.*,

Lithner, J. (2017). Principles for designing mathematical tasks that enhance imitative and creative reasoning. *Zdm*, *49*(6), 937–949.

Lockwood, E., DeJarnette, A. F., Asay, A., & Thomas, M. (2016). Algorithmic thinking: An initial characterization of computational thinking in mathematics. *North American Chapter of the International Group for the Psychology of Mathematics Education*.

Ludvigsen, S. (2015). *Fremtidens skole, fornyelse av fag og kompetanser*. Departementenes servicesenter.

Mason, R., Crick, T., Davenport, J. H., & Murphy, E. (2018). Language choice in introductory programming courses at Australasian and UK universities. Proceedings of the 49th ACM Technical Symposium on Computer Science Education,

McClain, K., & Cobb, P. (2001). An analysis of development of sociomathematical norms in one first-grade classroom. *Journal for Research in Mathematics Education*, *32*(3), 236-266.

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, *62*(2), 77-90.

Mercer, N. (2005). Sociocultural discourse analysis: Analysing classroom talk as a social mode of thinking. *Journal of Applied Linguistics and Professional Practice*, *1*(2), 137-168.

Merenluoto, K., & Lehtinen, E. (2004). Number concept and conceptual change: towards a systemic model of the processes of change. *Learning and Instruction*, *14*(5), 519-534.

Michaels, S., O'Connor, C., & Resnick, L. B. (2008). Deliberative discourse idealized and realized: Accountable talk in the classroom and in civic life. *Studies in philosophy and education*, *27*(4), 283-297.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. sage.

Misfeldt, M., & Ejsing-Duun, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. In K. Krainer & N. Vondrová (Eds.), *Proceedings of the Ninth Congress of the European Society for Research in Mathematics Education* (pp. 2524–2530). Charles University in Prague, Faculty of Education, and ERME.

Misfeldt, M., Jankvist, U. T., Geraniou, E., & Bråting, K. (2020). Relations between mathematics and programming in school: Juxtaposing three different cases. 10th ERME topic conference on mathematics education in the digital era, MEDA 2020, 16-18 September. Linz, Austria,

Moors, L., Luxton-Reilly, A., & Denny, P. (2018). Transitioning from block-based to text-based programming languages. 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE), Auckland, New Zealand.

Mozelius, P., Ulfenborg, M., & Persson, N. (2019). Teacher attitudes towards the integration of programming in middle school mathematics. *INTED 2019*, 701-706.

Nathan, M. J., & Knuth, E. J. (2003). A study of whole classroom mathematical discourse and teacher change. *Cognition and instruction*, *21*(2), 175-207.

Nelson, G., & Powell, S. R. (2018). A systematic review of longitudinal studies of mathematics difficulty. *Journal of Learning Disabilities*, *51*(6), 523-539.

NESH. (2016). *Forskningsetiske retningslinjer for samfunnsvitenskap, humaniora, juss og teologi*. https://www.etikkom.no/forskningsetiske-retningslinjer/Samfunnsvitenskap-jus-og-humaniora/

Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings: Learning cultures and computers* (Vol. 17). Springer Science & Business Media.

Oluwatayo, J. A. (2012). Validity and reliability issues in educational research. *Journal of educational and social research*, *2*(2), 391-391.

Pajankar, A. (2017). Introduction to Python. In *Python Unit Test Automation* (pp. 1–17). Apress.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Piaget, J. (1964). Part I: Cognitive development in children: Piaget development and learning. *Journal of Research in Science Teaching*, *2*(3), 176–186.

Piteira, M., & Costa, C. (2013). Learning computer programming: study of difficulties in learning programming. Proceedings of the 2013 International Conference on Information Systems and Design of Communication, Lisbon, Portugal.

Plomp, T. (2013). Educational design research: An introduction. In N. Nieveen & T. Plomp (Eds.), *Educational design research* (pp. 11-50). SLO.

Polya, G. (1957). *How to solve it: A new aspect of mathematical methods*. Prentice University Press.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*(5), 583–602.

Putnam, H. (1992). *Realism with a human face*. Harvard University Press.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1–24.

Rabardel, P. (2002). *People and technology: A cognitive approach to contemporary instruments.* https://hal.archives-ouvertes.fr/hal-01020705

Reiser, B. J. (2004). Scaffolding complex learning: The mechanisms of structuring and problematizing student work. *The journal of the learning sciences*, *13*(3), 273-304.

Resnick, L., Asterhan, C. S., & Clarke, S. (2017). Student discourse for learning. In G. E. Hall, D. M. Gollnick, & L. F. Qzinn (Eds.), *Handbook of teaching and learning.* . Wiley: Wiley-Blackwell.

Resnick, L. B., Bill, V., & Lesgold, S. (1992). Developing thinking abilities in arithmetic class. In A. Demetriou, M. Shayer, & A. Efklides (Eds.), *Neo-Piagetian theories of cognitive development: Implications and applications for education* (pp. 210-230). Routledge.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., & Silverman, B. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

Ryan, G. W., & Bernard, H. R. (2003). Techniques to identify themes. *Field methods*, *15*(1), 85-109.

Ryve, A. (2011). Discourse research in mathematics education: A critical evaluation of 108 journal articles. *Journal for Research in Mathematics Education*, *42*(2), 167–199.

Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129–141.

Schoenfeld, A. (1985). *Mathematical problem solving*. Academic Press.

Schoenfeld, A. (1985). Mathematical Problem Solving (1985). In: Academic Press, New York, NY.

Sevik, K. (2016). *Programmering i skolen*. Senter for IKT i utdanningen

Sfard, A. (2000). On reform movement and the limits of mathematical discourse. *Mathematical thinking and learning*, *2*(3), 157-189.

Silverman, D. (1985). *Qualitative methodology and sociology: describing the social world*. Gower Pub Co.

Skemp, R. (1986). The psychology of mathematics learning. *Suffolk: Penguin Books*.

Skemp, R. R. (1976). Relational understanding and instrumental understanding. *Mathematics Teaching*, *77*(1), 20–26.

Skolverket. (2017). *Få syn på digitaliseringen på grundskolenivå.* Retrieved from https://www.skolverket.se/publikationer?id=3783

Smit, J., & Van Eerde, H. (2011). A teacher's learning process in dual design research: Learning to scaffold language in a multilingual mathematics classroom. *Zdm*, *43*(6), 889-900.

Soloway, E. (1993). Should we teach students to program? *Communications of the ACM*, *36*(10), 21–24.
Star, J. R. (2005). Reconceptualizing procedural knowledge. *Journal for Research in Mathematics Education*, 404–411.
Stein, M. K., Engle, R. A., Smith, M. S., & Hughes, E. K. (2008). Orchestrating productive mathematical discussions: Five practices for helping teachers move beyond show and tell. *Mathematical thinking and learning*, *10*(4), 313-340.
Stein, M. K., Grover, B. W., & Henningsen, M. (1996). Building student capacity for mathematical thinking and reasoning: An analysis of mathematical tasks used in reform classrooms. *American Educational Research Journal*, *33*(2), 455–488.
Stein, M. K., & Smith, M. S. (1998). Mathematical tasks as a framework for reflection: From research to practice. *Mathematics Teaching in the Middle School*, *3*(4), 268–275.
Stephens, M. (2018). Embedding algorithmic thinking more clearly in the mathematics curriculum. *Proceedings of ICMI Study*, *24*, 483–490.
Stephens, M., & Kadijevich, D. M. (2020). Computational/algorithmic thinking. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 117–123). Springer.
Stillman, G. (2004). Strategies employed by upper secondary students for overcoming or exploiting conditions affecting accessibility of applications tasks. *Mathematics Education Research Journal*, *16*(1), 41–71.
Strauss, A., & Corbin, J. M. (1990). *Basics of qualitative research: Grounded theory procedures and techniques*. Sage Publications, Inc.
Sullivan, P., Clarke, D., & Clarke, B. (2013). *Teaching with tasks for effective mathematics learning* (Vol. 9). Springer.
Sullivan, P., Knott, L., & Yang, Y. (2015). The relationships between task design, anticipated pedagogies, and student learning. In A. Watson & M. Ohtani (Eds.), *Task design in mathematics education* (pp. 83-114). Springer, Cham.
Swan, M. (2013). Design-based Research in Mathematics Education. In *Encyclopedia of Mathemaitcs Education*. London: Springer.
Sweller, J. (2006). How the human cognitive system deals with complexity. In J. Elen & R. E. Clark (Eds.), *Handling complexity in learning environments: Theory and research* (pp. 13-26). Elsevier Science Ltd.
Szlávi, P., & Zsakó, L. (2006). Programming versus application. In M. R.T. (Ed.), *LNCS: 2nd International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (Vol. 4226, pp. 48-58). Springer.
Udir. (2006). *Læreplan i matematikk for realfag*. https://www.udir.no/kl06/MAT3-01/Hele/Timetall
Udir. (2019). *Nye læreplaner – grunnskolen og gjennomgående fag vgo*. https://www.udir.no/ Retrieved from https://www.udir.no/laring-og-trivsel/lareplanverket/Nye-lareplaner-i-grunnskolen-og-gjennomgaende-fag-vgo/
Vinnervik, P. (2021). *när lärare formar ett nytt ämnesinnehåll: Intentioner, förutsättningar och utmaningar med att införa programmering i skolan* Umeå universitet].
Vosniadou, S. (1994). Capturing and modeling the process of conceptual change. *Learning and Instruction*, *4*(1), 45-69.

Vosniadou, S. (2003). Exploring the relationships between conceptual change and intentional learning. In G. M. Sinatra & P. R. Pintrich (Eds.), *Intentional conceptual change* (pp. 377-406). Lawrence Erlbaum Associates.

Vygotsky, L. S. (1980). *Mind in society: The development of higher psychological processes*. Harvard university press. ((Original manuscripts [ca. 1930-1934]))

Wainer, J., & Xavier, E. C. (2018). A controlled experiment on Python vs C for an introductory programming course: Students' outcomes. *ACM Transactions on Computing Education (TOCE)*, *18*(3), 1-16.

Waits, B., & Demana, F. (2000). *Calculators in mathematics teaching and learning*. National Council of Teachers of Mathematics.

Watson, A., Ohtani, M., & Ainley, J. (2015). Task design in mathematics education. *Proceedings of ICMI Study*, *22*.

Watson, A., & Sullivan, P. (2008). Teachers learning about tasks and lessons. In D. Tirosh & T. Wood (Eds.), *International handbook of mathematics teacher education: Tools and Processes in Mathematics Teacher Education* (Vol. 2, pp. 109-134). Sense Publishers.

Weintrop, D., Bain, C., & Wilensky, U. (2017). Blocking Progress? Transitioning from Block-based to Text-based Programming. In: UCS Education.

Whiteley, W., & Mamolo, A. (2013). Optimizing through geometric reasoning supported by 3-D models: Visual representations of change. *Task Design in Mathematics Education. Proceedings of ICMI Study 22*, 129.

Wiliam, D., & Thompson, M. (2008). Integrating assessment with learning: What will it take to make it work? In C. A. Dwyer (Ed.), *The future of assessment: Shaping teaching and learning* (pp. 53–82). Lawrence Erlbaum Associates.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *Acm sigcse bulletin*, *28*(3), 17-22.

Yackel, E., & Cobb, P. (1996). Sociomathematical norms, argumentation, and autonomy in mathematics. *Journal for Research in Mathematics Education*, *27*(4), 458-477.

9 Appendices

A Solution to problem in 3.3

Important – this is not a mathematical proof, merely an explanation of the idea behind the solution.

The idea is to create a sequence of drops, with the maximum number of drops not changing no matter which floor is the solution. The first drop can, as a starting point, be anywhere. The second drop must then consider that we have used one drop already, and therefore needs to be selected such that the maximum number of drops does not change. To accomplish this, if the first floor chosen was at n, then the following floor needs to be $(n-1)$ floors up from the first floor. If unlucky, you need a maximum of n drops: a maximum of n if the first ball breaks on the first throw and a maximum of $(n-1)+1=n$ if the first ball breaks on the second floor chosen. For the third floor chosen, we need to choose floor number $n+(n-1)+(n-2)$ to keep the number of maximum drops constant. We will then have used $(n-2)+2=n$ drops

Since there is a maximum of 100 floors, we can build the following sequence and inequality (for $n \in \mathbb{N}$):

$$n+(n-1)+(n-2)+(n-3)+\cdots+1 \leq 100$$

Solving this for the greatest value gives $n=14$.

The sequence of floors should therefore be:

Floor you drop from	Most unlucky floor	Maximum number of drops
14	1	0+14
27	15	1+13
39	28	2+12
50	40	3+11
60	51	4+10
69	61	5+9
77	70	6+8
84	76	7+7
90	85	8+6
95	91	9+5
99	94	10+4
100	100	11

The maximum number of drops you need is 14 no matter how unlucky you are.

B MPP covering the bisectional method (in Norwegian)

Funksjoner 2 - Nullpunkter

Vi skal fortsette på programmet i **Funksjoner 1 - Plot** for å finne nullpunkter til en funksjon. Vi må først tenke gjennom hvordan vi skal gjøre dette. Vi vet stort sett hvordan vi skal finne nullpunkter hvis vi ser funksjonen, men vi bruker ikke alltid samme metode.

Oppgave 1

Diskuter med sidemannen hvordan (og om) man kan finne nullpunktet for alle funksjoner av typene under.

a) Alle andregradslikninger, for eksempel

$$f(x) = 2x^2 + 3x - 2$$

b) Alle tredjegradslikninger, for eksempel

$$g(x) = x^3 + 3x - 4$$

c) Alle rasjonale funksjoner, for eksempel

$$h(x) = \frac{2x + 3}{3 - x}$$

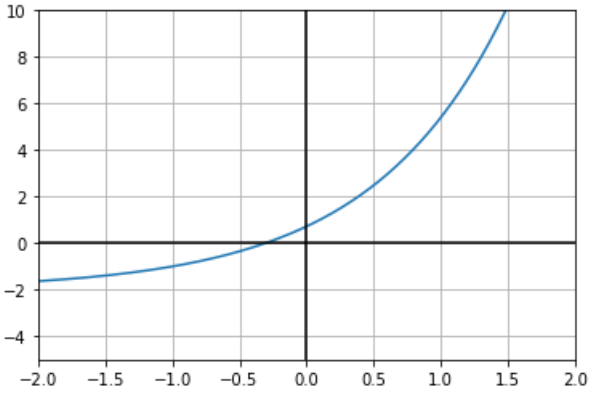
d) Alle eksponentielle funksjoner, for eksempel

$$i(x) = e^{2x+1} - 1$$

Det er flere eksempler, men metodene vi bruker varierer ganske mye fra funksjon til funksjon. Drømmen er å finne én metode som kan anvendes på alle funksjoner vi skriver inn. Vi skal se hvor langt på vei vi kan komme.

Starter med å se på funksjoner der funksjonsverdien skifter fortegn i nullpunktet, se eksempel under.

Figuren under viser en graf som er negativ til venstre for nullpunktet, og positiv til høyre for nullpunktet. Dette er det vi mener med at funksjonsverdien skifter fortegn i nullpunktet.



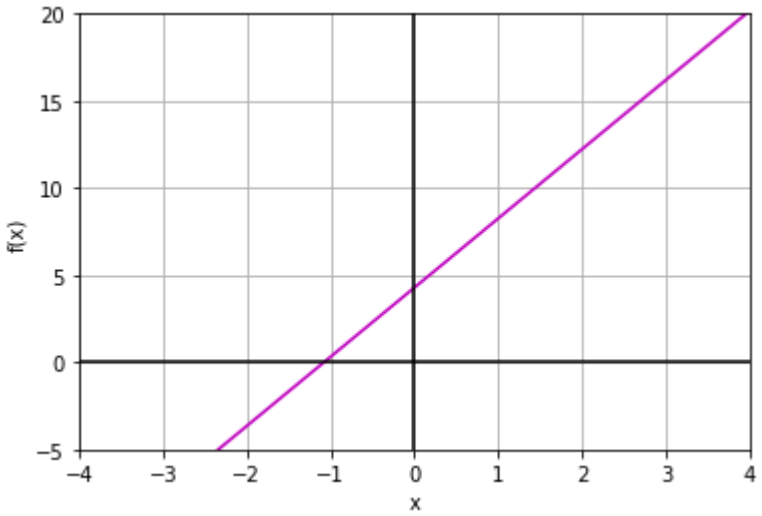
Vi må nå prøve å lage en metode for å finne nullpunktet. Så hvordan skal vi klare det?

Oppgave 2

a) Diskuter følgende spørsmål med sidemannen:

- Hva er felles for alle nullpunkter?
- Er det noen forskjeller på nullpunkter?

b) Tenk dere en maskin som kun kan gi deg funksjonsverdien når du oppgir en x-verdi. Alle andre spørsmål vil gi en feilmelding. Hvordan kan du systematisk gå frem for å finne nullpunktet til funksjonen under ved hjelp av denne maskinen? Maskinen kan regne så nøyaktig som du ønsker på funksjonsverdiene. Diskuter fremgangsmåter sammen med sidemannen.



c) En i gruppen får utdelt en del ark med grafer på. Kun én person i gruppen ser på arkene (og spiller maskinen). Resten av gruppen stiller spørsmål for å finne hvor grafen har nullpunktet sitt. Bruk metoden dere avtalte i forrige deloppgave. Sjekk om den fungerer på alle grafene dere fikk utdelt.

Halveringsmetoden:

Metoden avhenger av at vi vet i hvilket intervall vi skal lete. For å løse dette lar vi først programmet plotte grafen, for deretter å legge inn intervallet vi skal lete i.
La oss se på en funksjon fra en tidligere eksamen (H2017) som er vanskelig å finne nullpunktene til

$$f(x) = 2\ln(x^4 + 4) - \frac{1}{2}x$$

Oppgave 3

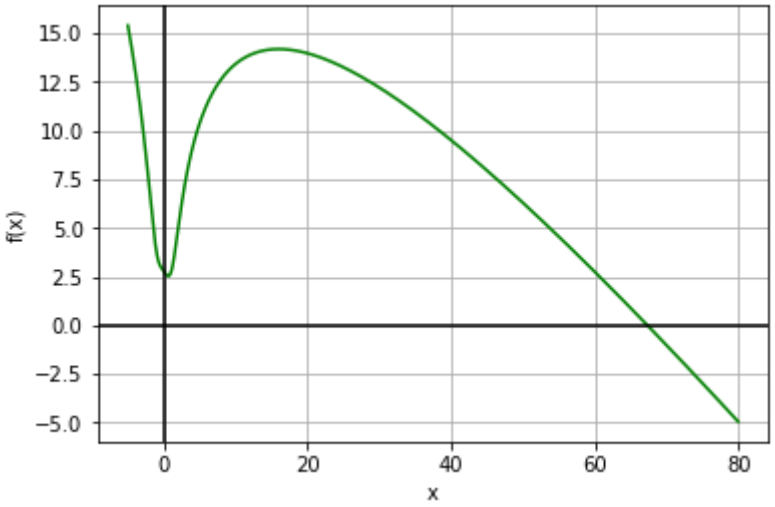
Prøv å finn nullpunktene ved regning. Klarer dere det? Hvor nærme kommer dere?

I oppgaven over fant dere at vi ikke kjenner til noen metoder for å løse denne typen oppgaver. Her må vi bruke det som kalles en numerisk fremgangsmåte (se side 12 for forklaring av numeriske metoder).

Fremgangsmåten for programmeringen presenteres i steg, der hvert steg forklarer en del av koden. Dere kan gjerne prøve selv først, men sørg for å likevel lese gjennom stegene slik at dere ser hva dere må ha med.

Steg 1 – Tegne grafen og lokalisere nullpunktene

Starter med å bruke programmet fra forrige time til å plotte grafen for å se hvordan den ser ut. Merk at i Python er `log()` kommandoen for den naturlige logaritmen. Etter å ha lekt litt med koden får man et plot som viser funksjonen. Vi ser også i hvilket område nullpunktet befinner seg. Grafen kan for eksempel se slik ut (dette er avhengig av hva du har valgt som `xlim` og `ylim`).



Fra denne grafen ser vi at funksjonen har ett nullpunkt i intervallet [60, 80]. Vi kan gjette på et mindre eller større intervall, men det har liten betydning for programmet. Intervallet [60, 80] er greit nok for at programmet skal vite hvor det skal lete.

Koden som gir grafen over er:

```
from pylab import *
x = linspace(-5, 80, 1000)   # Verdier for x

def funksjonsverdi(x):       # Funksjon som regner ut y-verdiene
    y = 2*log(x**4 + 4) - 0.5*x
    return y
y = funksjonsverdi(x)        # Kaller på funksjonen over

plot(x, y, 'g')              # Plotter x og y
grid()                       # Tegner rutenett
ylim(-5, 15)                 # Intervallet langs y-aksen
xlabel('x')                  # Navn x-akse
ylabel('f(x)')               # Navn y-akse
axhline(y=0, color='k')      # Tegner x-akse
axvline(x=0, color='k')      # Tegner y-akse
show()                       # Viser grafen
```

Steg 2 – Sette et intervall

Vi må først la programmet vite hvor det skal begynne å lete. Vi starter derfor med å spørre brukeren om hvor vi skal lete, se eksempel under Steg 3. Bruker her `input()` kommandoen. Siden intervallet alltid er et tall legger vi på kommandoen `float()` slik at programmet skjønner at det er et tall vi oppgir.

```
xv = float(input("x-verdi til venstre for nullpunkt: "))
xh = float(input("x-verdi til høyre for nullpunkt: "))
```

Steg 3 – Regne ut funksjonsverdiene

Vi regner ut funksjonsverdien til `xv` og `xh`. Hvis vi har oppgitt et korrekt intervall bør de ha forskjellig fortegn.

```
yv = funksjonsverdi(xv)      # y-verdien, venstre side
yh = funksjonsverdi(xh)      # y-verdien, høyre side

print("Venstre side: f(", xv, ") =", yv)
print("Høyre side: f(", xh, ") =", yh)
```

Eksempel:

Kjører programmet og setter `xv = 60` og `xh = 80.`

Ser at funksjonsverdiene har forskjellig fortegn.

Oppgave 4

Prøv programmet på funksjonene under. Bestem intervaller selv og sjekk at programmet gir forskjellig fortegn på hver side av nullpunktet.

a) $f(x) = 5 - 2x$

b) $g(x) = x^2 - 3x + 2$

c) $h(x) = e^{2x} - 4$

Hint: Et godt tips for å teste uten å slette det man allerede har skrevet er å sette en # foran. Da vil ikke programmet lese den delen av koden. Når man vil tilbake kan man bare ta vekk # igjen. Man kan også markere teksten man vil skal kommenteres og venstre-klikke og velge Comment/Uncomment (Ctrl + 1)

Steg 4 – Finne midtpunktet (halveringslinjen)

Etter å ha sjekket at *f* (xv) og *f* (xh) faktisk har forskjellig fortegn, sjekker vi funksjonsverdien til x-verdien i midten av intervallet, *f* (xm).

```
xm = (xv + xh) / 2            # Regner ut midtverdien til x
ym = funksjonsverdi(xm)       # y-verdien, midtpunkt

print("Midtpunkt: f(", xm, ") =", ym)
```

Fra eksempelet over så vi at *f* (xv) var negativ og *f* (xh) var positiv. Vi kjører programmet og får skjermbildet under.

Vi må nå tolke svarene vi har fått. Under er to påstander om svarene.

- Hvis $f(\text{xm})$ er negativ, vet vi at nullpunktet befinner seg mellom xv og xm
- Hvis $f(\text{xm})$ er positiv, vet vi at nullpunktet befinner seg mellom xm og xh

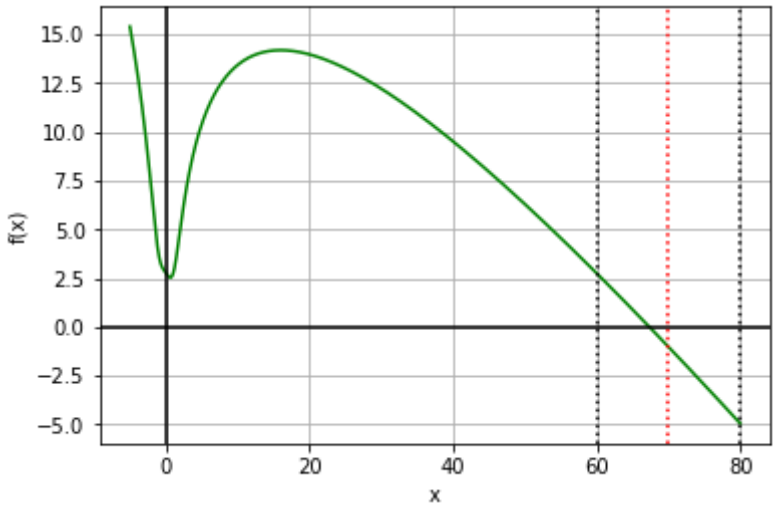
Oppgave 5

Diskuter med sidemannen hvorfor de to påstandene over må være korrekte.

Steg 5 – Sjekke fortegn og *if*-setning

Vi har sjekket fortegnet til `yv`, `ym`, og `yh`, og vi må nå avklare om nullpunktet ligger i intervallet [xv, xm] eller [xm, xh].

Figuren under viser linjene $x = 60$, $x = 80$, og linjen midt imellom (markert med rødt) $x = 70$. Vi ser at fortegnet til $f(80)$ er likt fortegnet til $f(70)$, og dermed vet vi at nullpunktet ligger i intervallet [60 , 70].



Vi må utvide konsekvensene av fortegnene til å gjelde generelt.

Oppgave 6

Bruk figuren over til å diskutere med sidemannen følgende påstander.

- Hvis `yv` og `ym` har samme fortegn, så vil nullpunktet ligge i intervallet [xm, xh]
- Hvis `yv` og `ym` har ulikt fortegn, så vil nullpunktet ligge i intervallet [xv, xm]

Vi ser at dette er en «hvis – så» prosedyre, og vi har lært om *if*-setninger, så dette virker lovende. Vi må oversette punktene over til en *if*-setning.

Oppgave 7

Hva er en rask måte å sjekke om to tall har samme eller motsatt fortegn ved hjelp av en *if*-setning? Diskuter med sidemannen. Etter å ha diskutert kan dere se en mulig løsning neste side.

En mulig løsning på oppgave 7:

```
if yv*ym < 0:
    # Mulighet 1: Her ligger nullpunktet mellom xv og xm
elif yv*ym > 0:
    # Mulighet 2: Her ligger nullpunktet mellom xm og xh
```

Vi må fortsatt finne ut hva vi skal gjøre for hver av de to mulighetene i kodebiten over.

Steg 6 – Metode og grafisk forklaring av halveringsmetoden

Metoden vi har kommet frem til er:

i) Finne midtpunkt i intervallet

ii) Sjekke funksjonsverdien for midtpunktet

iii) Kjøre *if*-setningen for å sjekke i hvilket av de to intervallene nullpunktet befinner seg

iv) Sette et nytt intervall

v) Starte på i) igjen

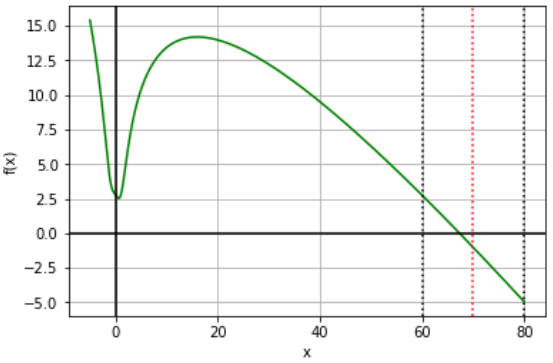
Metoden kan vi kjøre om igjen helt til vi er fornøyd med svaret vårt.

Oppgave 8

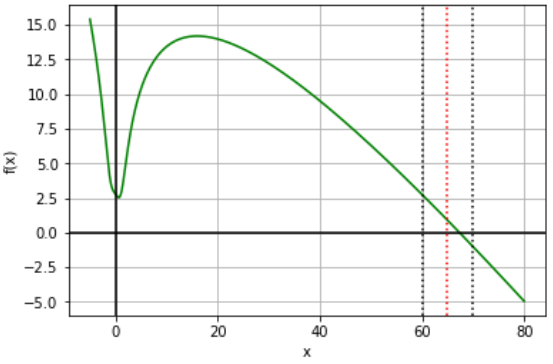
Når er vi fornøyd med svaret vårt? Diskuter med sidemannen

Viser fremgangsmåten med et par figurer på neste side.

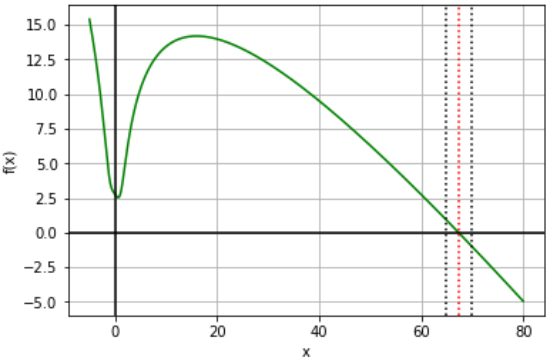
På figuren øverst har vi valgt x-verdiene 60 og 80. Deretter har vi funnet verdien midt imellom, her 70. Vi sjekker mellom hvilke verdier nullpunktet ligger, og finner at det ligger mellom punktet til venstre og midtpunktet.



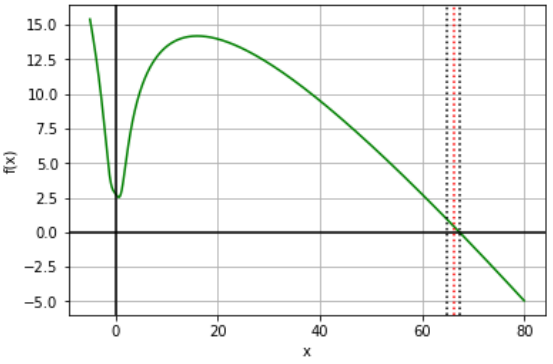
På neste figur tegner vi inn de nye avgrensende intervallet med to linjer (sort) og det nye midtpunktet (rødt). Vi sjekker igjen mellom hvilke verdier nullpunktet ligger, og finner at det ligger til høyre for midtpunktet.



Vi setter det nye intervallet vårt til linjene til høyre, og gjør prosessen om igjen.



Hvis vi fortsetter med dette vil vi komme til et punkt som er veldig nærme nullpunktet. Vi kan selv bestemme nøyaktigheten, men vi kan begrense nøyaktigheten til innenfor 2 desimaler for eksempel. Kan øke denne senere hvis man ønsker.

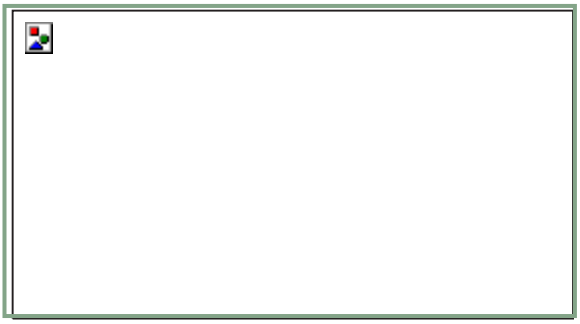


Steg 7 – Repetisjon om bruken av *while*-løkke (hopp over hvis dere husker)

Fra metoden over ser vi at vi må bruke en kommando som lar oss gjennomføre en prosedyre flere ganger inntil vi når den nøyaktigheten vi ønsker. En *while*-løkke egner seg godt her.
En *while*-løkke kjører en prosess inntil en betingelse er oppfylt. En kort *while*-løkke er som følger:

```
i = 1                       # Setter i til verdien 1
while i < 10:               # Mens i er mindre enn 10
    print("While-løkken har nå kjørt", i, "ganger.")
    i = i + 1               # Øker i med en hver gang
print("Der fikk i verdien", i, ", så løkken er slutt")
```

Skriver ut til skjerm:



Oppgave 9

a) Hva skjer når du skriver `i + 2` istedenfor `i + 1`?

b) Hva skjer når du endrer `i < 10` til `i < 15`?

c) Hva skjer når du endrer `i = 1` til `i = 20`?

d) Kan du endre *while*-løkken slik at den skriver ut de ti første negative tallene?

Steg 8 – Programmere *if*-setningen ferdig

Starter med å se på *if*-setningene våre og utvikler dem videre.

```
if yv*ym < 0:
    # Mulighet 1: Her ligger nullpunktet mellom xv og xm
elif yv*ym > 0:
    # Mulighet 2: Her ligger nullpunktet mellom xm og xh
```

Vi starter med at første mulighet er sann. Da er det nye intervallet [xv, xm]. Vi setter da at den nye høyre-verdien (`xh`) er lik `xm`. Siden `xv` ikke har endret seg gjør vi ingen endring på den.

```
if yv*ym < 0:
    # Mulighet 1: Her ligger nullpunktet mellom xv og xm
    xh = xm
```

For å vite mer nøyaktig hvor nullpunktet er trenger vi det nye midtpunktet til intervallet. Vi må regne ut den nye verdien til `xm`.

```
if yv*ym < 0:
    # Mulighet 1: Her ligger nullpunktet mellom xv og xm
    xh = xm
    xm = (xv + xh) / 2
```

Det siste vi må gjøre er å regne ut de nye funksjonsverdiene til disse tre punktene som avgrenser det nye intervallet. Vi kan hoppe over `yv` siden den er ikke endret.

```
if yv*ym < 0:
    # Mulighet 1: Her ligger nullpunktet mellom xv og xm
    xh = xm                      # Setter ny høyre x-verdi
    xm = (xv + xh) / 2           # Regner ut ny midtverdi
      ym = funksjonsverdi(xm)    # Regner ut ny funksjonsverdi
      yh = funksjonsverdi(xh)    # Regner ut ny funksjonsverdi
```

Da er vi ferdige med den ene halvdelen av if-setningen.

Oppgave 10

Den andre delen av *if*-setningen er rimelig lik, se om du og sidemannen kan lage den før dere ser på en mulig løsning på neste side.

```
if yv*ym < 0:
    # Mulighet 1: Her ligger nullpunktet mellom xv og xm
    xh = xm                      # Setter ny høyre x-verdi
    xm = (xv + xh) / 2           # Regner ut ny midtverdi
      ym = funksjonsverdi(xm)    # Regner ut ny funksjonsverdi
      yh = funksjonsverdi(xh)    # Regner ut ny funksjonsverdi
elif yv*ym > 0:
    # Mulighet 2: Her ligger nullpunktet mellom xm og xh
    xv = xm                      # Setter ny venstre x-verdi
    xm = (xv + xh) / 2           # Regner ut ny midtverdi
      ym = funksjonsverdi(xm)    # Regner ut ny funksjonsverdi
      yv = funksjonsverdi(xv)    # Regner ut ny funksjonsverdi
```

Steg 9 – Programmere inn while-løkken

Siste steg i programkoden er heldigvis rimelig greit. Det eneste vi nå må bestemme er hvor nøyaktig vi ønsker at svaret vårt skal være. La oss starte med at vi ønsker å ha svaret med en nøyaktighet på to desimaler. Vi kan da si at hvis avstanden mellom xv og xh er mindre enn 0,001 så er vi fornøyd. Siden vi setter nøyaktigheten til andre desimal, runder vi av svaret til to desimaler med kommandoen `round(tall, antall_desimaler)`

```
while xh - xv > 0.001:   # Løkken kjører så lenge dette er
                         # oppfylt
      if yv*ym < 0:
        # Mulighet 1: Nullpunktet mellom xv og xm
        xh = xm                  # Setter ny høyre x-verdi
        xm = (xv + xh) / 2       # Regner ut ny midtverdi
        ym = funksjonsverdi(xm)  # Ny funksjonsverdi
        yh = funksjonsverdi(xh)  # Ny funksjonsverdi
      elif yv*ym > 0:
        # Mulighet 2: Nullpunktet mellom xm og xh
        xv = xm                  # Setter ny venstre x-verdi
        xm = (xv + xh) / 2       # Regner ut ny midtverdi
        ym = funksjonsverdi(xm)  # Ny funksjonsverdi
        yv = funksjonsverdi(xv)  # Ny funksjonsverdi
print("Nullpunktet er ved x =", round(xm, 2))
```

Oppgave 11

Hvorfor velger vi 0,001 som nøyaktighet når vi skal vise svaret med kun to desimaler? Kunne vi ikke valgt en nøyaktighet på 0,01? Eller burde vi kanskje valgt 0,0001? Diskuter med sidemannen.

Steg 10 – Kjøre programmet

Da kan vi prøve å kjøre programmet og se hva som skjer.

Dette ser rimelig ut, ut ifra grafen.

Oppgave 12

a) Sett inn og sjekk at denne x-verdien faktisk *er* et nullpunkt for funksjonen vi startet med.

b) Prøv å endre nøyaktigheten til nullpunktet. Når er du fornøyd? Når burde man være fornøyd? Er det noen ganger det er viktig å ikke være for presis med svaret? Diskuter med sidemannen

Steg 11 – Numeriske metoder

Numeriske metoder er en fremgangsmåte der man bruker en algoritme (prosedyre) for å beregne eller løse matematiske problemer. Denne algoritmen er numerisk som betyr at for hver gang vi kjører algoritmen så kommer vi (ideelt sett) til et svar som er bedre/mer nøyaktig enn det forrige. Merk at dette er kun i et idealisert tilfelle. Det eksisterer begrensninger i programmering når for eksempel antallet desimaler vokser. Motsetningen til numeriske metoder er symbolske (og analytiske) metoder. Eksempelvis kan man løse likningen $x^2 = 3$ analytisk og få svaret $\pm\sqrt{3}$. Ved numerisk beregning vil svaret være et avrundet desimaltall for eksempel $\pm 1{,}732051$ (ved seks desimalers nøyaktighet).
Nøyaktigheten kan vi ofte bestemme, men ofte vil 3-4 desimaler være godt nok. Per 2017 var den numeriske nøyaktigheten av π beregnet til mer enn 10^{13} desimaler.

Steg 12 – Kommentar til slutt

Helt til slutt er det viktig med en kort kommentar angående programmet vi nå har laget. Programmet vil ikke klare å finne nullpunkter der funksjonen har et topp- eller bunnpunkt i nullpunktet. Grunnen til dette er at funksjonsverdien ikke skifter fortegn når funksjonen krysser nullpunktet. Det er likevel metoder for å løse dette, se oppgave 13.

Oppgave 13

Kan dere diskutere med sidemannen hvordan vi kan gå frem for å lage en metode som inkluderer å finne nullpunktene beskrevet over?

10 Articles

Article 1
Morten Munthe and Margrethe Naalsund
Designing mathematical programming problems
Submitted to Digital Experiences in Mathematics Education – under 2nd review

Article 2
Morten Munthe
Facilitating exploratory talk through mathematical programming problems
Submitted to NOMAD

Article 3
Morten Munthe
Programming in the mathematics classroom – The types of adversities students encounter
Accepted, to be printed autumn of 2022 in Acta Didactica Norden

Designing mathematical programming problems

Abstract

The growing use of programming in mathematics classrooms presents a challenge linked to implementation in general, and task design in particular. This article presents design ideas for mathematical problems incorporating programming in which the focus remains mainly on learning mathematics and less on learning programming. The article starts by reviewing the theoretical background for technology implementation and design then presents the methodology for the design before exploring and discussing the design ideas with an in-depth example. Building on the idea of adidactical situations from the theory of didactical situations, the design illustrates a possible way of implementing programming in the mathematics classroom to facilitate mathematical learning.

Keywords: task design, mathematical learning, text-based programming, algorithmic thinking, adidactical situations

Declarations

The work is funded by the Municipality Education Authority, City of Oslo, Norway, and the Research Council of Norway. On behalf of all authors, the corresponding author states that there is no conflict of interest.

Data availability statements

The datasets generated and analysed during the current study are not publicly available due the fact that they constitute an excerpt of research in progress but are available from the corresponding author on reasonable request.

1 Introduction

The increase in access to computer technology in mathematics classrooms over the last decades has been significant (Ran et al., 2020), allowing students to, through technology, engage in mathematical activities involving modelling (Greefrath and Siller, 2017), geometry (Sinclair et al., 2016), problem solving (Psycharis and Kallia, 2017), and more (Bray and Tangney, 2017; Drijvers, 2015). Reviews of the effect of technology on mathematical learning are mixed. While the 2015 OECD-study argues that there is little evidence that more use of computers leads to positive effects in mathematical achievement (OECD, 2015) several studies before and after this study suggest that technology-enhanced mathematical instruction is an effective means to support mathematics achievement (Cheung and Slavin, 2013; Li and Ma, 2010; Rakes et al., 2010; Young, 2017). The technological evolution and corresponding experiences developed by students, teachers and researchers, will continuously influence mathematics classrooms. "Educational technology is not a homogeneous "intervention" but a broad variety of modalities, tools, and strategies for learning. Its effectiveness, therefore, depends on how well it helps teachers and students achieve the desired instructional goals" (Ross, 2010, p. 19). We need more research on how the various forms of technology can promote the different aspects of mathematical learning.

Several countries have implemented programming as a part of their national curriculum, and programming is often combined with mathematics (Balanskat and Engelhardt, 2015; Bocconi et al., 2018). The implementation of programming in mathematics is not a new idea. In the 1980s, Papert (1980) argued for the use of programming to facilitate student learning. The implementation did not succeed, partly due to an inability to apply programming in mathematics as a learning resource (Misfeldt and Ejsing-Duun, 2015). There is now a resurgence of research investigating the implementation of programming in the mathematics classrooms in schools with many aspects of this change currently being investigated (e.g. Benton et al., 2017; Psycharis and Kallia, 2017). One aspect of the implementation of programming in the mathematics classroom is task design, which is the focus for this article. The research and discussion of the relationship between task design and mathematics learning has a long tradition (Arbaugh and Brown, 2005; Krainer, 1993; Watson et al., 2015), and is a core research area in mathematics education (Sierpinska, 2004). Task design was, however, not a focus for research until the mid-1970s (Wittmann, 1995), and while "*didactical design has always played an important role in the field of mathematics education, […] it has not always been a major theme of theoretical interest in the community*" (Artigue, 2009, p. 7). Task

design is now increasingly more prominent in the mathematics education research community (Watson et al., 2015, p. 27). Leung and Bolite-Frant (2015, p. 4) define *mathematics task design* as designing activities situated in pedagogical environments that provide boundaries within which students engage in doing mathematics, leading to the construction of mathematical knowledge. Additionally, the advent of technology in the mathematics classroom has caused research to delve into the use of technology in task design and learning. With the extensive research into both task design and, in particular, technology task design, where programming would reside, one might expect investigation into the combination of programming and mathematics at the upper secondary school level. In reality, however, the existing research on this combination is sparse, but growing (Benton et al., 2018; Bråting et al., 2020; Forsström and Kaufmann, 2018; Heintz et al., 2017; Psycharis and Kallia, 2017). The current research indicates that programming can contribute to problem-solving and mathematical learning (Psycharis and Kallia, 2017), but as it is still in its infancy more research is needed.

As tasks are the main component of the "things to do" in the mathematical classroom (Watson et al., 2015) and with the advent of programming in the mathematics classroom, there is a need for research into how to design tasks that combine programming and mathematics. The present article aims to *present and discuss design ideas for problems implementing programming as a tool for learning mathematics*. First, we present a theoretical background on the implementation of programming in relation to mathematical thinking and learning, secondly on task design in mathematics education linked to technology. Thereafter, we propose and discuss the design ideas and how they were developed together with an example of such a design and a short observation of student responses. Finally, we discuss the design and the choices surrounding the design.

2 Programming and mathematical learning

Algorithmic thinking is a highly relevant concept studying programming as a tool for learning mathematics. According to Stephens and Kadijevich (2020), algorithmic thinking is based on three cornerstones: *decomposition*, *abstraction* and *algorithmisation*. Decomposition is about breaking the problem apart into smaller pieces to be solved individually. Abstraction refers to the process where learners identify the most relevant information needed to solve the problem and eliminate extraneous details (Dreyfus, 2020) and is defined by Skemp (1986) as an activity by which we become aware of similarities. Algorithmisation reflects the development of converting a process or a set of processes into an algorithm. Lockwood et al. (2016) links algorithmic thinking to the development of deep procedural knowledge defined by Star (2005) as 'knowledge of procedures that is associated with comprehension, flexibility, and critical judgement', while Abramovich (2015) links it to the development of conceptual knowledge.

Learning in mathematics can be seen as 'the construction of a web of connections – between classes of problems, mathematical objects and relationships, real entities and personal situation-specific experiences' (Noss and Hoyles, 1996), which is closely related to the process of abstraction (Mitchelmore and White, 2007). This follows along the lines of other studies using terms such as *relational understanding* (Skemp, 1976) and *conceptual knowledge* (Hiebert, 2013). A relational understanding means that the learner knows both what to do and why – that is, there is conscious reasoning behind the mathematical actions taken. Conceptual knowledge is rich in relationships, and the connected web of units of knowledge is achieved by constructing relationships between the available pieces of information. The combining factor is that they all recognise that a student gains understanding through the development of a network interlinking mathematical concepts, where he or she can draw upon several concepts and connections between concepts to undertake a problem. Mathematical learning is an evolving process and mathematical understanding is the mental connections between facts, procedures and ideas (Hiebert, 2013). These relationships are revised when new information is difficult to assimilate or when previous relationships are inadequate to explain a new problem (Piaget, 1964; Skemp, 1976). Thus, mathematical thinking and reasoning is a highly integrated part of learning mathematics with an in-depth understanding (Kilpatrick et al., 2001; Lithner, 2017). Reasoning comprises the capacity to think logically about the relationships among concepts and situations through reflection, explanation and justification (Kilpatrick et al., 2001). This view of mathematical learning influences the kind of tasks one believes will contribute to such learning. Problem solving is important to stimulate mathematical thinking and has been a focus for research for a long time (Polya, 1957; Schoenfeld, 1985, 2013). To a much larger degree than the more commonly used routine-based exercises (e.g. solving numerous equations based on given examples), genuine problems comprise the idea that the students must figure out a method rather than being given a method to achieve the unknown solution. Problem solving therefore has rich potential for stimulating deeper learning skills, as described above, and is closely connected to algorithmic thinking.

Through the use of programming as a tool in the mathematics classroom, algorithmic thinking can be advantageous for mathematical learning, because decomposition, abstraction and algorithmisation are key ingredients applicable to many problems of mathematical content, such as problem solving (Polya, 1957; Schoenfeld, 2013) and mathematical reasoning (Kilpatrick et al., 2001; Lithner, 2017). Stephens (2018) points out that more 'attention to algorithmic thinking in schools could help students expand their problem-solving techniques and to explain and justify their mathematical reasoning' (pp. 489-490).

3 Technology-based task design in mathematics

The design of mathematical tasks can be divided into several categories, ranging from open- versus close-ended tasks, to contextualised versus non-contextualised tasks, to routine-based versus cognitively demanding tasks (Berisha and Bytyqi, 2020; Stein et al., 1996). Stein et al. (1996, p. 426) state that 'tasks used in mathematics classrooms highly influence the kinds of thinking processes in which students engage, which, in turn, influences student learning outcome'. This, in turn, influences the design of tasks, including those using programming as a tool.

Designing tasks that use digital technologies is complex and difficult (Joubert, 2007; Laborde and Sträßer, 2010) partly because of the ability of the computer to perform hidden or unknown mathematical procedures referred to as 'black box' by Buchberger (1990). In comparison, a 'white (or transparent) box' is characterised by the students being conscious of the mathematics they ask the tool to perform. Rabardel (2002) also uses the concepts of 'black box' and 'glass box' to refer to the dimension of the operative transparency of the technological tool being used.

A review of research in computer education strongly indicates that learning to program is difficult as "students exhibit various misconceptions and other difficulties in syntactic knowledge, conceptual knowledge, and strategic knowledge" (Qian and Lehman, 2017, p. 17). Syntactic knowledge is the understanding of the building of the code, such as the use of parenthesis, equals signs, and semicolons. Conceptual knowledge is the students' model of code executions, such as variables, if-statements, and the sequential execution of the code. Strategic knowledge is the process of planning, writing, and debugging programs. Text-based programming also creates additional challenges for students, such as focusing on syntax rather than the mathematical meaning of the code (Lewis, 2010; Resnick et al., 2009).

According to Brousseau (1997), valuable mathematical learning will more likely take place when the students are committed to a problem situation. In the theory of didactical situations (TDS), knowledge is a property of a system consisting of a subject and a *milieu*. The problem, together with the programming language and the students of a particular group, creates the *milieu*, or the environment, with which they interact. The interaction can be conceptualised as the conversation within and the feedback from the milieu (Brousseau, 1997).

A core concept in TDS is the *didactical contract* that exists between the educator and the students of the milieu. The contract implies a set of expectations regarding mathematical knowledge and the responsibilities of both the teaching and the learning process. The students expect that working through the set of problems provided by the educator, they will learn the required mathematics, and the educator expects the students to complete the problems through the given guidelines. As the students are working with the problems, they are interacting with the *milieu*, which can be both collaborative and antagonistic. This period of interaction is the *adidactical situation*, where the students have the initiative and the responsibility for the outcome of the learning process. The adidactical situation persists for a time because the students know through the didactical contract that they would not receive a problem they were unable to solve or learn from. Preceding an adidactical situation is often a *didactical situation*, in which the educator or the task itself offers a problem to be solved. Succeeding the adidactical situation is another didactical situation where the educator or the task links the knowledge gained to the aim of the problem. We argue that both the preceding and succeeding didactical situation do not depend solely on the educator but can be parts of the designed task. This is especially true of the succeeding didactical situation, in which the problem design can facilitate students discussing the implications of the solution of the task in a broader context. The educator's ability to recognise and interpret the students' actions into a system of what they ought to learn is important and is called the *institutionalisation* of the acquired knowledge (Brousseau, 2008). It is possible for the task to contain and facilitate institutionalisation through explanatory text or visuals connecting the actions to a system of learning. The aim of TDS is to design situations that facilitate the construction of knowledge by the student, in which the student takes responsibility as a participant in the problem-solving process. The task design should facilitate a situation whereby by solving the task the students gain the desired targeted knowledge.

Allowing the students to adapt strategies to reach the desired target knowledge is challenging, and Brousseau (1997)suggests that they will not succeed unless facilitated by the task. If the students encounter and overcome 'obstacles' as they are working through the given tasks, an adaptation may take place. Brousseau (1997, p. 83) defines an *epistemological obstacle* a form of knowledge that has been relevant and successful in particular contexts, often including school contexts, but that at some moments becomes false or insufficient. Obstacles can also be of a different nature: *ontogenic obstacles*, for instance, relate to the limitation of the students and lack of required prior learning, and *didactical obstacles* relate to the presentation of the subject, 'the result of narrow or faulty instruction' (Harel and Sowder, 2005, p. 34). Both ontogenic and didactical obstacles inhibit learning and should be avoided (Brousseau, 1997; Harel and Sowder, 2005), while epistemological obstacles can promote learning. An epistemological obstacle is explained by Balacheff (1990, p. 264) as follows 'Any content has to be supported by the pupils' previous knowledge. But this old knowledge can turn into an obstacle to the constitution of new conceptions, even though it is a necessary foundation. But more often than not, to overcome this obstacle is part of the construction of the meaning of the new piece.' This explanation indicates that task design needs to build on previous knowledge to create new conceptions and facilitate the construction of new meaning by designing for adidactical situations. When students are working on tasks, they will experience periods when they know how to perform the required actions, and periods when they encounter obstacles. The adidactical situations will allow the students to reconsider their strategies, develop new pathways, discuss with their peers, conjecture and experiment, which are all related to the intended learning process (Leung and Baccaglini-Frank, 2016). Thus, adidactical situations stimulate mathematical thinking and reasoning, and develop conceptual knowledge (Hiebert, 2013; Skemp, 1976) and hence lead to the acquisition of deep learning of the required mathematical content (Noss and Hoyles, 1996).

Ideally, the milieu should 'provide feedback that moves the learner forward' (Wiliam and Thompson, 2008, p. 15). Feedback can come from the task, the group conversations, or the program. The task facilitates learning by generating elements and accompanying actions that the students undertake together with the feedback provided by the milieu. Feedback should enable the students to evaluate meaningful strategies, which attest to the building of new knowledge (Artigue et al., 2014). In task design, including technological tools, feedback can be particularly important; see Bokhove and Drijvers (2010). The feedback helps the students to construct knowledge by becoming engaged in the solving of the problem and by refining their concepts and strategies (Brousseau, 1997). Such cognitive development, in the TDS framework, is part of the adidactical situation. A property of technological tools is, to a varying degree, the provision of valuable feedback to the user. Feedback from a computer is 'quick and essentially unlimited…at "no cost"' (Hillel, 1992, p. 209). This allows task designers to facilitate a greater range of experimentation and verification of ideas. Three things – the ability to use technology to produce results, immediate feedback, and novel ways of looking at mathematical objects – could support a change in the nature of communication in mathematical problem solving (Drijvers et al., 2016). The change creates a challenge for designers, as tasks ideally allow for experimentation, exploration, and discussions. Several studies present guidelines, criteria and/or principles for designing good mathematical problems (e.g. Johnson et al., 2017; Kieran, 2019; Sullivan et al., 2012; Wiliam and Thompson, 2008), advocating focus on conceptual, open-ended tasks, linking technical and theoretical activity, integrating questions calling upon pattern seeking, and applying technological tools for generating and testing conjectures.

4 The design

With the ambition of facilitating in-depth learning (Kilpatrick et al., 2001; Lithner, 2017; Skemp, 1976) through algorithmic thinking (Stephens and Kadijevich, 2020), the problems in this project were designed based on the belief that the adidactical situation consisting of the milieu, feedback from the milieu, and the didactical contract between students and educator (Brousseau, 1997) constitutes the most important part of the learning process. Here, we will describe and discuss the design ideas for tasks using programming as a tool for learning mathematics. Emphasising the importance of problems and problem solving for in-depth learning (as described in Section 2), 'problem design' is preferred instead of the more common 'task design'.

In the following section we present and discuss the design ideas for the problem type of this article, emphasising the idea of obstacles and with a focus on implementing text-based programming[1] as a tool for learning mathematics before elaborating on the design process.

4.1 Mathematical programming problems

Our problem design incorporates text-based programming as a tool for learning mathematics and therefore needs to satisfy several criteria. The problem design should be appropriate with respect to the content of the mathematics curriculum, including a clear visualisation of the mathematics underlying the programming. It should also facilitate student interaction with the milieu by creating adidactical situations. To create an adidactical situation, we present students with a problem they are initially incapable of solving satisfactorily with their current knowledge. In their search for alternative strategies, the intended new knowledge can be acquired by overcoming the problem. The design of the problem facilitates the search for alternative strategies. Each problem the students encounter contains a discussion task, forcing them to receive feedback from the milieu through conversation with the group. The specification of the problem should initially illuminate the inadequacy of the existing methods available to the students before enabling them to develop a more general or powerful solution (Ruthven et al., 2009). Applying these ideas, *mathematical programming problems* were designed.

The mathematical programming problems (MPP) consists of a series of problems designed to facilitate the students discovering a new concept or a new connection through adidactical situations. The structure of an MPP is to start by recalling previous mathematical knowledge before exposing a limitation of this knowledge through an example. The students are then given the responsibility to investigate and discuss how to resolve the limitation through a set of problems. Each problem allows for the progression towards the target knowledge of the MPP, which is a successful solution method built on understanding important mathematical concepts. MPPs consist of two parts: the first is without programming, and the second with programming.

The first part consists of non-programming tasks asking the students to recall previously known mathematical concepts needed to progress through the MPP. There are two main reasons for this. First, it ensures that all students have the opportunity to recall the same information, which is applicable and beneficial (Stillman, 2004) throughout the problem, especially later in the MPP when complexity increases. Second, since the first task is a recollection of previously known concepts, it has a low threshold for completion and facilitates students' ability to complete the initial part of the task. This upholds the part of the didactical contract between students and educators, where the students expect to be able to resolve the problem with their knowledge. Later, when the complexity increases, the recollected knowledge facilitates the students in solving the problem. The problems continuously build upon the recalled knowledge to facilitate mathematical learning and attaining the intended knowledge. The MPP can be viewed as a process in which each new problem facilitates an adidactical situation aiding the students in their progression towards the desired knowledge. Problems can also directly or indirectly reveal inadequacies or challenges within mathematics that can, to a lesser or greater extent, be solved through programming.

The programming part starts in a similar way, recalling a set of previously known programming procedures necessary to start building the required program. As the students work through the different problems of the MPP, they are continuously changing and developing the program further. Each section of code that is possible to execute represents a verifiable step towards the desired knowledge. Verification can be done by inspecting graphs, output data, or some other form of visual or readable response. The various sections of the problem design should facilitate an increased transparency of the problem, in which the students become conscious and engaged in the mathematical knowledge that they apply through the programming process. The programming part of the problem design should be based on the need to balance the additional complexity programming brings to the mathematics classroom with the overall design of the problem. The complexity in terms of the lower versus higher level of demands of the problem (Stein and Smith, 1998), together with the students' previous knowledge and self-efficacy, determine whether they will succeed or not in solving the given problem (Hoffman and Spatariu, 2008). The maintenance of high-level (cognitive) demands is dependent on several factors, including thinking and reasoning, self-monitoring, building on previous knowledge (Stein and Smith, 1998), and the link between the problem and the aim throughout the MPP. Soloway (1993) argues that during the programming process, the learner uses powerful problem solving and thinking strategies. Students first need to solve a problem

[1] Text-based programs are typed using a keyboard, follows a set of rules for the specific programming language and are stored as text files (See Figure 9 for an example).

mathematically, followed by reflecting on how to express the solution through computer programming (Papert, 1980; Szlávi and Zsakó, 2006).

The aim of an MPP is to facilitate the students discovering one or more sets of target knowledge by applying mathematical knowledge and solving a set of problems. The path through a particular problem can be quick, as in solving a simple equation or performing a calculation, or time-consuming when several sequential procedures must be handled.

An obstacle can vary in complexity with the intention to create an antagonistic milieu, which the students struggle to resolve. Whether an obstacle will initiate a successful problem-solving process is dependent on the knowledge of the students, which is linked to the milieu. An example is the question 'What is the area between the x-axis and the graph of a given quadratic function on a particular domain?', which, unless the student has learned integration, is a challenging problem. The problem design can also include changes in the students' understanding of a known concept by introducing new information, where there may be difficulties in incorporating the associated new concepts. An example is the development of their understanding of the numbers starting from the set of integers, the expansion to the set of rational numbers, the further expansion to the set of real numbers, and the final expansion to the set of complex numbers. The tool (here, the programming language) the students are using can also cause difficulties in MPP, contributing to additional complexity (Ko et al., 2004). This spans over a wide range, from writing code and reviewing errors to misunderstanding the output generated by the program. An example of an epistemological obstacle is initially failing to understand the right answer may be 'no solutions' when searching for the real roots of a polynomial of even degree by a calculator or a computer algebra system (CAS). Programming in itself can create an additional ontogenic obstacle for the students, as they are not experienced with the stringent rules of programming. The students are used to the need for accurate inputs in CAS, but the type of input that they know often consists of one line, not the typically large number of lines often required in a program. The MPP aims to limit this ontogenic or didactical obstacle through giving the students skeletal code (Figure 9) and several means, as will be discussed later. Problem design should facilitate obstacles to an extent that the students need to struggle (Hiebert, 1984; NCTM, 2014) but do not lose faith in the didactical contract, generating an adidactical situation. If the struggle continues for an extended period, the students might lose this faith and end up giving up without completing the MPP.

The result of the students taking the initiative and responsibility for the outcome of a designed mathematical task, should be an increased network between concepts or the creation of a new concept, which coincides with the target knowledge of the MPP. Asking for the possible solutions to quadratic equations is contributing towards the resolution and facilitates later asking the students to create a program that both solves and plots a quadratic equation that has no real solution by visualising the connection between the crossings of the x-axis and the number of solutions. This insight increases the network of mathematical concepts and facilitates learning. The adidactical situations contribute to the students' progress in the intended learning trajectory. This is not necessarily a leap in understanding but a stepping-stone for consolidating an existing concept or introducing a new concept based on existing knowledge, which later builds the foundation for another target knowledge. The build of the MPP is summarized into seven phases below, where each phase is exemplified with concrete problems from the MPP concerning the bisectional method, followed by a discussion of each phase of the design in section 5.

1. **Recalling relevant mathematical knowledge**

   Enabling the students to recall the mathematical knowledge required to initiate the progress through the MPP and their search for the target knowledge. This can limit the amount of ontogenic obstacles.

   In Figure 1 this consist of the calculation and discussion of properties of zero-point for different functions. This allows the students to recollect properties of the zero-point and method for finding zero-points, which will be the base for the target knowledge.

Discuss with your group how (and if) one can find the zero-points for all the function-types below.

a) All quadratic equations, for instance

$$f(x) = 2x^2 + 3x - 2$$

b) All third-degree polynomials, for instance

$$g(x) = x^3 + 3x - 4$$

c) All rational functions, for instance

$$h(x) = \frac{2x + 3}{3 - x}$$

d) All exponential functions, for instance

$$i(x) = e^{2x+1} - 1$$

What are the differences and similarities between the method used to find the zero-points in these functions?

Figure 1: Problem from phase 1 of the MPP concerning he bisectional method.

2. Presenting an inadequacy or challenge in the mathematical method.

The start of a problem, where presenting the students with one or more examples of when a mathematical method becomes false or inadequate. This forces the students to search for new strategies, as they, through the didactical contract, are confident that a solution and learning opportunity exists.

In Figure 2 this consist of the search to find zero-points for a general and unknown continuous function covering a range of both negative and positive function values. This problem allows the students to use and apply their knowledge of zero-points to a new type of problem. The strategy they start to develop here will be used later in the same MPP.

Imagine a machine that contains and unknown linear function and will give you a function value when you input a numerical x-value. All non-numerical inputs will yield an error.

How can you systematically use this machine to find the zero-point for the unknown function to the right using this machine? What if the function is non-linear?

The machine can be as accurate as you would like concerning the function-value. Discuss possible strategies with your group.

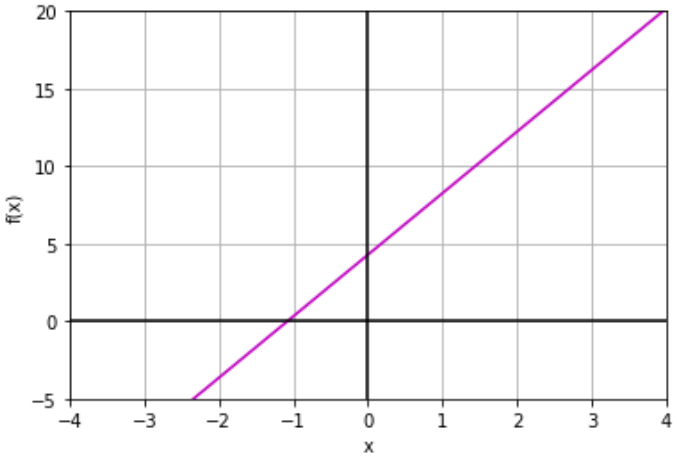


Figure 2: Problem from phase 2 of the MPP concerning the bisectional method.

3. **Creating a mathematical strategy for applying programming to the inadequacy or challenge**

Selected problems initiate a process of combining existing mathematical knowledge with new strategies or concepts. This further builds towards the target knowledge where the students through the adidactical situation can develop new pathways through interaction with the milieu. Phase 3 should facilitate exploration, discussion, and evaluation allowing the students to assemble their existing knowledge into a new strategy or method.

The problem in Figure 3 consists of developing strategies to numerically approximate the zero-points through interacting with the milieu. The problem facilitates the students discussing and testing different strategies building towards the target knowledge.

Elect one group member to take the role of the machine from the previous problem. The «machine»-students collects a set of graphs from the teacher, where the zero-point is marked with an accuracy of six decimal places. The «machine»-students can only reply with function-values, when given x-values from the other group members. The task is to find the zero-point(s) of the unknown graphs as efficiently as possible. You may discuss strategies with everyone in the group, including the «machine»-student.

Figure 3: Problem from phase 3 of the MPP concerning the bisectional method.

4. **Recalling programming commands and structures**

Enabling the students to recall the programming knowledge required to initiate the progress through the MPP and their search for the target knowledge. This can limit the amount of ontogenic obstacles.

In the presented MPP this consist of plotting, implementing elementary calculations, the use of functions, and the structure of while-loops. Figure 4 is a typical code the students wrote to recall how to plot (line 10-17) and how to build (line 3-5) and call (line 8) a function. The result after running the code is shown in Figure 5.

```
1   from pylab import *                 # Importing required commands
2
3   def function_value(x):              # Function calculating the y-values
4       y = 2*log(x**4 + 4) - 0.5*x
5       return y
6
7   x = linspace(-5, 80, 1000)          # Values for x
8   y = function_value(x)               # Calling the function above
9
10  plot(x, y, 'g')                     # Plotting x and y
11  grid()                              # Draw a  grid
12  ylim(-5, 15)                        # Sets the y-interval
13  xlabel('x')                         # Labels the x-axis
14  ylabel('f(x)')                      # Labels the y-axis
15  axhline(y = 0, color = 'k')         # Draws the x-axis
16  axhline(y = 0, color = 'k')         # Draws the y-axis
17  show()                              # Shows the graph
```

Figure 4: Code-snippet from phase 4 of the MPP concerning the bisectional method.

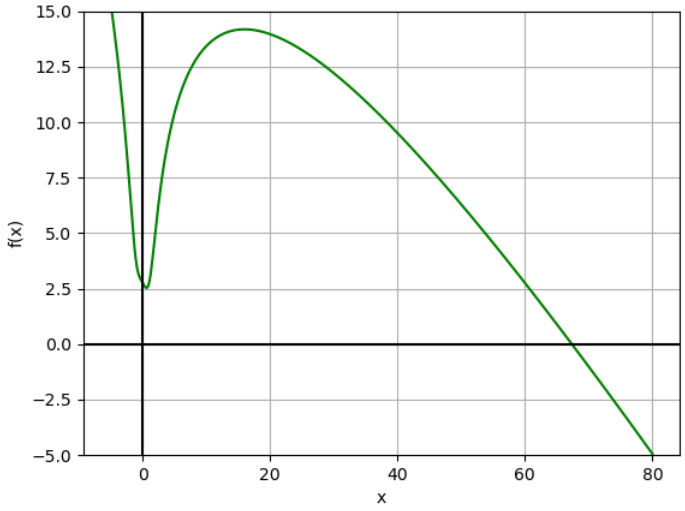


Figure 5: Result from running the code in Figure 4

5. Decomposing the mathematical strategy into programmable segments

Using programming as a tool, the students now implement mathematical strategies to create a basic program facilitating explorative search and analysis. Programming facilitates algorithmic thinking to decompose the mathematical strategy into smaller programming sections and implement it in the program.

In the presented MPP this consist of decomposing the strategy into smaller, programmable mathematical segments. As an example, the students' need to input two starting x-values (xl and xr) where the function values (yl and yr) have different signs for the program test for the middle value and initiate the numerical procedure (Figure 6 and Figure 7).

```
20    xl = float(input("x-value left of the zero-point: "))
21    xr = float(input("x-value right of the zero-point: "))
22
23    yl = function_value(xl)                  # Function value, left side
24    yr = function_value(xr)                  # Function value, right side
25
26    print("Left side: f(", xl, ") =", yl)    # Prints the function value
27    print("Right side: f(", xr, ") =", yr)   # Prints the function value
```

Figure 6: Code-snippet from phase 5 of the MPP concerning the bisectional method.

```
x-value left of the zero-point: 60

x-value right of the zero-point: 80
Left side: f( 60.0 ) = 2.754757115060663
Right side: f( 80.0 ) = -4.943786727296455
```

Figure 7: Result from running the code in Figure 6, where the inputs are 60 and 80.

After calculating the middle-value (xm) between the two x-values and the corresponding function-value (ym), another part of the decomposed strategy is to build a segment that check the sign of the function-value.

What is a quick way to check if two numbers have the same or opposite signs with the help of an *if*-statement? Discuss with your group.

Figure 8: Problem from phase 5 of the MPP concerning the bisectional method.

```
32  if yl * ym < 0:
33      # Possibility 1: Zero-point located between xl and xm
34  elif yr * ym < 0:
35      # Possibility 2: Zero-point located between xm and xr
```

Figure 9: Code snippet and example of skeletal code as a solution to problem in Figure 7

6. **Composing the program through assembling the decomposed segments.**

   The students build the program by applying algorithmic thinking to the decomposed segments of the mathematical strategy resulting in both abstraction and algorithmization. The composing of the program implements the mathematical strategy to resolve the inadequacy or challenge.

   In Figure 10 this consist of combining all programming segment into a completed code. Part of this completed code below illustrates a while-loop numerically calculating the zero-point. This code is built upon the code in Figure 9.

```
39  while abs(xr - xl) > 0.001: # Set accuracy of zero-point value
40      if yl * ym < 0:
41              # Possibility 1: Zero-point located between xl and xm
42              xr = xm                  # Updates right x-value
43              yr = function_value(xr)  # New right function-value
44      elif yr * ym < 0:
45              # Possibility 2: Zero-point located between xr and xm
46              xl = xm                  # Updates left x-value
47              yl = function_value (xl) # New left function-value
48
49      xm = (xv + xh) / 2               # New middle value
50      ym = function_value (xm)         # New middle function-value
51
52  print("The zero-point is x =", round(xm, 2))
```

Figure 10: Code snippet from phase 6, where several elements (from Figure 4 and Figure 6 (line 43 and 47), Figure 9 (line 40 and 44)) are combined into a code segment.

7. **Exploring limitations and affordances of the program through a mathematical lens.**

   After the creation of a program, there is a need for evaluation. Revising and discussing the affordances of the program allows for consolidation of the knowledge throughout the MPP. Investigating the limitations of the program facilitates open-ended mathematical exploration and discussion.

   In Figure 11 this consist of discussing the limitations of what functions the numerical bisectional method applies to.

Are there any functions where the program is unable to calculate the zero-point? Explore and discuss with your group.

Figure 11: Problem from phase 7 of the MPP concerning the bisectional method.

A visual representation of the design ideas, emphasising the adidactical situations in the MPP, is presented in Figure 12.

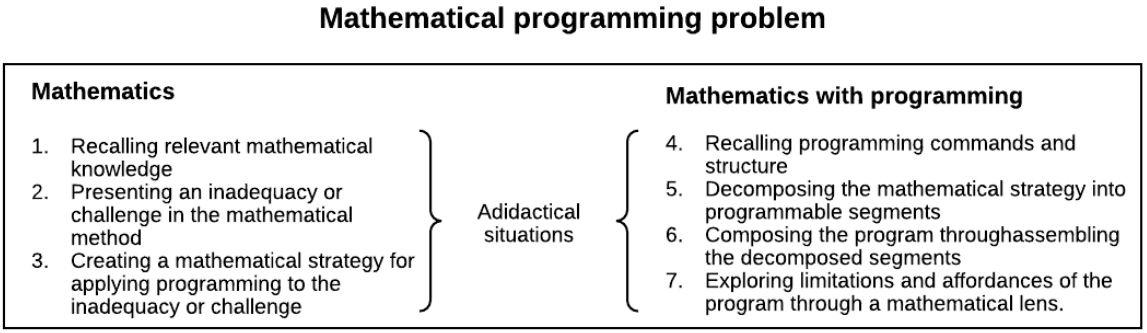


Figure 12: A visual representation of the design ideas of the MPP

4.2 Developing the design ideas

The development of the problems uses elements from *didactical engineering* (DE) as a basis for implementing the design ideas (Brousseau, 2008). DE is a valuable approach for creating and evaluating the design of problems and consists of several phases, elements of which we adopt in this study: a preliminary analysis, conception and a priori analysis, realisation, observation and data collection and, finally, an a posteriori analysis and validation. The methodology of DE aims at creating *situations* in which the target mathematical knowledge provides a successful solution to the given problem and the students reach this solution through interaction with the milieu. The role of the educator is to facilitate *devolution*, in which the students accept the mathematical responsibility of solving the problem and thereby develop an adidactic interaction with the milieu. An a priori analysis was not performed, however an iterative analysis over several implementations were performed, where the initial design was based on personal experiences from both teacher and researcher. The latter three elements of DE are used, where realisation is presented at the beginning of section 4, and observation and an a posteriori analysis are presented in section 5. The design and its conception originate from evaluating the *didactic variables* influencing the interaction between the mathematical content and the milieu. The didactic variables consist of linking programming as a tool to learning mathematics, considering the added difficulty that programming brings to mathematics, enabling students to discuss mathematics and facilitating mathematical learning. "These variables condition the milieu, thus the interactions between students and knowledge, the interactions between students and between students and teachers, thus the exact opportunities that students have to learn, how and what they can learn" (Artigue, 2015, p. 5).

The first MPP was implemented in a mathematics classroom for students in the second to last year of secondary school (age 17) with some basic knowledge of programming acquired from a 10-hour crash course in Python-programming. An initial design concerning quadratic equations was generated and implemented, together with an observation regime. The MPP was given to the students, and they worked on the problem for one lesson (90 minutes). Both the responses from the students in the form of video recordings and logs and the logs from the researcher initiated a revision of the development of the MPPs. The first iteration only had a short description of what the students were to program, such as "build a program that solves the quadratic equation" and "evaluate the number of solutions to the quadratic equation using the program you have created". During the implementation, the students spent most of their time struggling with the programming and not discussing or applying mathematical knowledge, and therefore several students failed to acquire the intended target knowledge. The first iteration was still valuable as it revealed several changes that could be made to the design. Following the analysis, an iterative revision process, in which the design of the MPP was modified to better fit with the design ideas, were undertaken. The MPP was changed to include more skeletal code and facilitating students' exploration, discussion, and discovery of the mathematical target knowledge. When the interaction with the milieu comprised mathematical discussions, both the students and the researchers reported it to be valuable, especially when the discussion explicitly connected programming issues to mathematics. The final design explored the quadratic equation and the number and types of possible solutions, focusing on the benefits of utilising programming. The students were to create a program that took the coefficients of a quadratic equation as the input and produced as output a message

stating the corresponding number of roots and the values of these roots if they existed. The aim of the design was to apply programming to a mathematical context that was familiar to the students, preventing ontogenic or didactical obstacles, and enable them to create a program initiating discussions between the students (creating feedback from the milieu). The discussions focused on the number of solutions and on how to prevent the program from performing invalid calculations. Communication and collaboration are essential aspects in order to deepen the students' conceptual and procedural knowledge, in which negotiating, providing arguments and considering various perspectives are crucial interactions for learning (Woo and Reeves, 2006; Yackel, 2002). Learning occurs in a classroom situation where the students' meet tasks that promote discussing mathematics, questioning results, applying known mathematical insight to unfamiliar situations and reasoning in the exploration and solving of challenging mathematical ideas and problems (Drijvers et al., 2016). The experiences from this first iteration went into the design of a series of new MPPs, one of which is presented above.

5 Discussing the design ideas

This section will present one MPP that aims to introduce students to numerical methods in mathematics by applying the numerical bisectional method to find zero-points of continuous functions. The MPP consists of the problems shown in section 4 in addition to several others described here. The students have previously learned an intuitive explanation of the intermediate value theorem, forming the mathematical basis for this MPP. Numerical mathematics is suited for the implementation of programming, as it utilises one of the greatest advantages of programming, namely repeatedly performing a particular calculation many times. The students have never used or encountered numerical methods previously in the curriculum.

The reason for choosing this method was to expose the widespread misunderstanding that all mathematical questions have both a distinct method for solving and an exact solution. This erroneous perception permeates classical school mathematics but is challenged by counter examples requiring numerical methods to approximate the zero-value. Additionally, this function is a gateway into justifying the use of programming to extract properties of functions not solvable by classical school mathematics.

The overarching design idea of the MPP is creating adidactical situations facilitating mathematical learning opportunities for the students. These adidactical situations will be discussed in the following sub-sections, which are organised according to the seven steps (Figure 12) – the design ideas. The discussion will highlight how the design of the MPP can promote mathematical learning and how the students responded.

5.1 Recalling relevant mathematical knowledge

The first problem (Figure 1) asks the students to discuss how to find the zero-point for a selection of given continuous functions (quadratic, cubic, exponential, and rational), followed by discussing the properties of zero-points in general. The aim of the first part of the task is for the students to recall previous knowledge regarding zero-points of functions, including methods for solving and the number of solutions. The second part of the task builds on this with a discussion of different types of zero-points. The aim is for the students to propose different properties of zero-points. The task is open ended, and the students answered everything from a simple statement that all zero-points touch the x-axis to more complex properties, such as whether the method is useful for finding extremal- or turning points.

The recollection of previous mathematical knowledge enabling the student to build a scaffold for the rest of the MPP, which is an important prerequisite for constructing relationships between the required pieces of the target knowledge (Hiebert, 2013; Noss and Hoyles, 1996; Skemp, 1976). Even though the problem formulation is identical, namely 'solve for the zero-point(s)', the method for each function varies greatly. More function-types could be added, but for the purpose of this problem, four different types were chosen. Giving the students chosen examples of functions allows for a discussion of mathematical concepts, such as valid solutions for the rational function and the existence of zero-points for variations of exponential functions, building conceptual knowledge (Hiebert, 2013; Noss and Hoyles, 1996). Asking the students to 'solve for some zero-point' facilitates problem-initiated feedback, applying the general solution to a specific example. The feedback is intended to come from the milieu, and primarily from the discussion within the group.

As the students had previously worked with properties of different functions, this problem created few challenges for the students. The latter two functions (c and d, Figure 1) particularity created a discussion involving valid solutions (c) and dependencies for the function to have a zero-point (d).

5.2 Presenting an inadequacy or challenge in the mathematical method.

The problem (Figure 2) does not involve programming or the use of the word bisectional method but introduces the mathematical concept by a simplified example in the form of a game. Formulating the question as a strategic game will engage the students (McGonigal, 2011; Zichermann and Linder, 2010). It also promotes their conceptual knowledge (Hiebert, 2013; Noss and Hoyles, 1996) and algorithmic thinking (Stephens and Kadijevich, 2020). This is due to the need for both the strategic development and the corresponding repeated checking and verification of the evolving strategy. Presenting the problem as open, the students themselves can propose ideas and strategize to find what they consider the best approach. The group discussion provides feedback, which can be checked against the given graph. The aim is for the students to apply their knowledge in a new and not previously experienced way, as an initial guess at a strategy.

The students discussed several different strategies here, from a chaotic trial and error of different x-values to a much more systematic guessing of x-values. As the graph in the problem (Figure 2) displayed a linear function, many students proposed that finding two values was enough, arguing that two points allowed the function to be calculated and thereby finding the zero-point. The question “What if the function is non-linear?” was added to avoid the discussion to strand there. This small change made the students continue their discussion.

5.3 Creating a mathematical strategy for applying programming to the inadequacy or challenge

The problem (Figure 3) allows for further development of the strategy by electing one student from each group to simulate the machine. This is important in the abstraction-, and algorithmisation-parts of algorithmic thinking (Stephens and Kadijevich, 2020). The goal of the problem is for the students to verify and/or modify the zero-finding strategy, arriving at an effective method for approximating the zero-point(s). Such reflections and evaluations are important aspects of algorithmic thinking (Stephens and Kadijevich, 2020) and thus important for the development of conceptual knowledge (Abramovich, 2015). The choice to display the function graph with the zero-point was to focus on the strategy on an unknown expression were calculating the zero-point is impossible. The number of decimal places of the zero-points was to avoid the students finding the zero-points instantly. With six decimal places the students may continuously guess bringing them closer and closer to the ’correct‘ value, including a discussion of what the correct value may be. The machine-simulating student provides feedback to the group and may participate in their search for a strategy. Students controlling the structure of the task reduces possible obstacles and builds an adidactical situation While the machine-simulating student is an obstacle blocking the solution, he is also a resource for the group in recognising the possibilities and limitations of the method they are working on. Two of the function graphs included in the game have more than one zero-point, which facilitates interesting questions such as ‘How do we know that we have found all the zero-points?’, which scaffold a later discussion of limitations of the program (phase 7, section 5.7). The trial-and-error aspect of the problem has characteristics like a computer game in which different inputs elicit different responses and the ’game‘ is to discover an effective strategy. This makes the task more attractive to students (McGonigal, 2011; Zichermann and Linder, 2010), engaging them in completing the problem.

The students were very active in this part of the MPP, where suggestions and discussions were represented in every group. The average time the students spent on this problem was about 20-25 minutes, consisting of both the game and the following discussion. The choice to elect a “machine”-student was successful, allowing the discussion to move forward even when they encountered obstacles, as the "machine”-student could assist in their search for both the zero-point and an effective strategy. Discussions included locations of the zero-point after a couple of x-values had been suggested, the kind of function it could be (given that only the “machine”-student saw the graph), and some groups even discussed if there were any way to know if there were more than one zero-point besides guessing over a wide range of x-values. This all contributes to what Brousseau (1997) calls valuable mathematical learning since the students are committed to the problem situation. Asking the students how closely they can approximate a zero-point leads to the idea of *estimation* and numerical evaluation instead of finding an exact solution. The aim is to get the students to evaluate different starting values for testing, spurring reasoning and algorithmic thinking (Kilpatrick et al., 2001; Stephens and Kadijevich, 2020).

5.4 Recalling previous programming knowledge

The programming part of the MPP (Figure 6) assists the students in preparation for building a computer program that will eventually estimate the zero-point to an accuracy defined by the students themselves. This step facilitates programming as a tool for learning mathematics through emphasising all three cornerstones of algorithmic

thinking (Stephens and Kadijevich, 2020), elaborated throughout the remaining sections. The students build a code that plots the graph of a function and explore it for several functions. The function in figure 4 (line 4, $f(x) = \ln(x^4 + 4) - x$) was chosen as while a solution exists, it is not possible to express it exactly. The function was used throughout the rest of the MPP as a test function for the students.

If they struggle, a skeleton code (like a template) was given, as some lines of code (i.e. the plot code in Figure 4 (line 13-17)) are not critical to understanding mathematics and could become an obstacle. This idea also follows the design emphasising that mathematics is the primary focus and that the programming is a tool to facilitate mathematical learning. The students are given freedom to alter the program, which contributes to their sense of ownership of both the code and the resulting mathematical calculations. The problem encourages the students to try for themselves first, as this builds their ownership of both the resulting computer program and their learning (Chan et al., 2014).

Previous versions of this problem did not contain any skeleton-code, which caused many students to seek help from the teacher as the adidactical situation did not come to fruition. With the skeleton code in place, most students completed the problem without any outside assistance since the milieu consisting of the group discussion and the problem itself was enough to maintain the adidactical situation (Brousseau, 1997). Some students completed the problem without help, while others relied on reviewing previously created code or asking the teacher or group for help.

5.5 Decomposing the mathematical strategy into programmable segments

Here the students need to decompose (Stephens and Kadijevich, 2020) the strategy they have developed so far into smaller segments that are possible to transform into programming code. There are a lot of smaller pieces needed to complete the code, where the first (Figure 3) is to initialise the strategy the students reached when they located the zero-point in the game-like problem (phase 3). The decomposition and transformation into code segments is a central aspect in the algorithmic thinking (Stephens and Kadijevich, 2020) when solving an MPP. The decomposition consists of several separate segments such as inputting a starting value to the left and right of the zero-point, finding the middle value between two values, checking the sign of the function value to decide the interval in which the zero-point is located, and finally create a loop that continues this process until a zero-point with the desired accuracy is reached. While certain problems directly relates to the decomposition (figure 6), other are left to the students, such as the code segment calculating the middle value. The problem is open as the group discusses how to decompose their strategy and build the necessary code pieces.

Initially, the students had problems decomposing their strategy and needed assistance in proceeding, but after being shown an example of a decomposed piece (figure 6); they were able to proceed discussing the decomposition of their strategy. This part of the MPP facilitated a lot of discussion regarding both the decomposition and the mathematical aspects behind the decomposition. The students spent time discussing how to check for opposite signs, since this was partly in conflict with their normal way of doing mathematics. If done by hand, they would not need to create an algorithm as it would be obvious from looking at the numerical value, but how would they create an algorithm such that a program would decide.

5.6 Composing the program through assembling the decomposed segments.

Here the students combine their deconstructed pieces of mathematical programming code to a finished program. They are in essence reconstructing their strategy with the individual pieces from phase 6.

To do so, they need to understand the mathematical concepts as well as how to use programming to solve a mathematical challenge. The relationship between programming and mathematics emerges in this phase of the problem-solving process. Through the previous problems, the students, having built the strategy and created the individual decomposed programming pieces now must complete the computer program. The loop-constructions in programming (*for*- and *while*-loops) can be linked to the systematic algorithmic processes in mathematics for calculating a sequence of improved approximations for the zero-point. The students can adjust smaller code segments and test them out instantly, allowing for exploration and the initial testing of the code.

The students did display a few difficulties assembling the code, where the most common obstacle was sequencing the code segments from phase 5. The obstacle was resolved quickly as the students were able to copy and paste code segments around quickly resolving the issue. The students were quickly moving on to the initial testing of the code, making sure that it worked for the presented examples (figure 10), essentially moving to phase 7. As the students had worked through the previous problems, they were not displaying frustration during this phase, rather

they were showing signs of eagerness to complete the program and start testing it out. Several students tried running the program several times before they had finished the entire code but were not discouraged when they received errors during this phase of the MPP.

5.7 Exploring limitations and affordances of the program through a mathematical lens

The problem facilitates a discussion regarding the accuracy of approximate solutions, which is lacking in upper secondary school mathematics, in which almost every mathematical task has an answer given in either exact form or rounded to a few decimal places. The students are rarely, if ever, asked about the exactness of a solution for a given case; this is often left to the other sciences, such as physics and chemistry. When the students have decided on the required accuracy of the zero-point, they implemented the requirement into the code using a *while*-loop (Figure 10). The final problem (Figure 11) is to verify the program for several other functions and starting guesses and reveal the limitations of the program. This is, for mathematical learning, a highly important phase of the problem-solving process (Polya, 1957; Schoenfeld, 1985). Additionally, 'When does the program *fail* to approximate the zero-point of a function?'. Does it depend on the guesses, the nature of the function, or something else? The final question of MPPs should preferably include such an open-ended question and could be followed by an additional fruitful question: 'Can we avoid the program failing to approximate the zero-point of a function and if so, how?' The purpose of the last question is to encourage exploration and evaluation, which is important for in-depth learning (Kilpatrick et al., 2001; Lithner, 2017). The last question also brings algorithmic thinking and mathematical learning intimately together by allowing the students to discuss their programming solution, including its inherent algorithm. They need to understand and explain why some values (or functions) work and others do not, what causes the program to fail to find some zero-point and, finally, how to apply both mathematics and programming to resolve the required problem. The MPP ends with an explanation of the bisectional method, linking the target knowledge to the MPP. To ensure that everyone grasps the link, the educator can also conduct this towards the end of the lesson.

The students were very eager to test out their program. When asked why they were so eager they responded that they wanted to make sure that the program they had created worked for the function types they knew and the challenge to find functions or starting values that did not work. Every student discovered that they had to make sure to input correct starting values, but few students thought to explore function where the zero-point and extremes overlap. This specific case was presented by the teacher at the end of the lesson as an example of when the program does work.

The structure of the MPP, from the initial recollections of previous knowledge regarding the zero-point of known simple functions, through the challenge of finding the zero-point for more advanced functions, to the numerical mathematical solution and the implementation of the numerical method into a programming environment, has facilitated the conceptual knowledge of numerical methods for approximating the solutions of mathematical problems.

6 Concluding thoughts and implications

This article has presented design ideas for what we have called mathematical programming problems, a series of problems that resides within TDS and didactical engineering. The aim is to promote in-depth mathematical learning (Hiebert, 2013; Noss and Hoyles, 1996; Skemp, 1976) through using programming as a tool in the learning process. The design of the MPPs consists of seven phases facilitating the students' interaction with the milieu. From the recollection of knowledge to the development of new strategies, the MPPs facilitate the target knowledge using programming as a tool. When students are working on the MPP, they will experience periods when they know what to do and periods when they encounter obstacles facilitating mathematical learning opportunities. The obstacles, when becoming an epistemological obstacle, allows the students to rethink their strategy, develop new pathways, discuss with their peers, conjecture, and experiment, which are all related to the intended learning (Leung and Baccaglini-Frank, 2016). Thus, the MPP has the potential to spur mathematical thinking and reasoning and to develop conceptual knowledge (Hiebert, 2013; Skemp, 1976), and hence deep learning of the mathematical content (Noss and Hoyles, 1996).

We have explored these design ideas through a numerical mathematics example, in which challenging the students to apply their existing knowledge of zero-points to new and unfamiliar functions, allowing them to re-evaluate

their strategies, was central. The *epistemological obstacle* occurs when the relevant knowledge of zero-points that have previously been successful, but now is simply inadequate and they need to apply their knowledge in a new way. The feedback and interaction with the milieu creating the adidactical situation allows for overcoming the obstacle, which Balacheff (1990) links to the construction of meaning. In the presented MPP, the students encountered functions in which they were unable to find the zero-point by using known methods. , Through a series of problems, they found alternative strategies for locating the zero-point aided by the milieu consisting of the problem, and feedback from running the programming code and their peers. All through the creation of the program, the students engaged in the adidactical situation, allowing them to apply their existing knowledge in a new way. The result is where the students hopefully have reached the target knowledge and learned new mathematical concepts.

The balance in designing an MPP is delicate for several reasons. The problems need to have their roots deeply in the mathematics curriculum while taking advantage of the possibilities that programming brings. One challenge is the choice of where and how to implement programming in the mathematics classroom. There is a real risk of going overboard and applying programming to areas where other tools are more strategic, undermining the value of programming as a mathematical tool. There are examples where coding is less beneficial for enhancing learning (Hayes and Stewart, 2016; Kalelioglu and Gülbahar, 2014), and as (Popat and Starkey, 2019) writes, 'if the academic aim is for students to learn mathematical problem solving, teaching these skills directly is more effective than learning these through coding.' Programming does have advantages that other technologies, such as GeoGebra and Computer Algebra Systems (CAS), does not have. Transparency, where the students have to input the code and therefore every calculation is visible unlike CAS where it is often a "black box". Simulating complex problems, for instance Monty Carlo simulations within probability. Numerical calculations (as shown here), where both the transparency of the method and the capacity to numerically calculate the zero-point to a great accuracy is present. Finally, building one's own tool creates ownership and understanding of both the program and the mathematical method. This last statement needs to be investigated further, but from the small sample presented, there are indications to support the statement.

The design presented investigates the complexity of, and progress through, a mathematical programming problem to learn mathematics using programming. The balance between obstacles can yield an MPP that can facilitate mathematical learning and algorithmic thinking with the use of programming as a tool. Further empirical research investigating the implementation of MPPs will build towards creating a set of design principles for the use of programming as a valuable tool in mathematics education.

7 References

Abramovich, S. (2015). Mathematical problem posing as a link between algorithmic thinking and conceptual knowledge. *Teaching of Mathematics, 18*(2).

Arbaugh, F., & Brown, C. A. (2005). Analyzing mathematical tasks: A catalyst for change? *Journal of Mathematics Teacher Education, 8*(6), 499–536.

Artigue, M. (2009). Didactical design in mathematics education. *Nordic research in mathematics education: Proceedings from NORMA08 Copenhagen*.

Artigue, M. (2015). Perspectives on design research: The case of didactical engineering. In *Approaches to qualitative research in mathematics education* (pp. 467–496). Springer.

Artigue, M., Haspekian, M., & Corblin-Lenfant, A. (2014). Introduction to the theory of didactical situations (TDS). In *Networking of theories as a research practice in mathematics education* (pp. 47–65). Springer.

Balacheff, N. (1990). Towards a problématique for research on mathematics teaching. *Journal for Research in Mathematics Education*, 258–272.

Balanskat, A., & Engelhardt, K. (2015). Computing our future. Computer programming coding. Priorities, school curricula initiatives across Europe. In. European Schoolnet, Brussels.

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education, 3*(2), 115–138.

Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International journal of child-computer interaction, 16*, 68-76.

Berisha, V., & Bytyqi, R. (2020). Types of mathematical tasks used in secondary classroom instruction. *International Journal of Evaluation and Research in Education, 9*(3), 751–758.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). The Nordic approach to introducing Computational Thinking and programming in compulsory education. *Report prepared for the Nordic@ BETT2018 Steering Group*, 397-400.

Bokhove, C., & Drijvers, P. (2010). Digital tools for algebra education: Criteria and evaluation. *International Journal of Computers for Mathematical Learning, 15*(1), 45–62.

Bray, A., & Tangney, B. (2017). Technology usage in mathematics education research–A systematic review of recent trends. *Computers & Education, 114*, 255–273.

Brousseau, G. (1997). *Theory of didactical situations in mathematics*. Kluwer Academic Publishers.

Brousseau, G. (2008). Research in mathematics education. In M. Niss (Ed.). *Proceedings of the 10th international congress on mathematical education*, 244–254.

Bråting, K., Kilhamn, C., & Rolandsson, L. (2020). Integrating programming in Swedish school mathematics: description of a research project. MADIF12: the twelfth research seminar of the Swedish Society for Research in Mathematics Education, 14-15 Jan 2020, Linnaeus University, Växjö, Sweden,

Buchberger, B. (1990). Should students learn integration rules? *ACM Sigsam Bulletin, 24*(1), 10–17.

Chan, P. E., Graham-Day, K. J., Ressa, V. A., Peters, M. T., & Konrad, M. (2014). Beyond involvement: Promoting student ownership of learning in classrooms. *Intervention in School and Clinic, 50*(2), 105-113.

Cheung, A. C., & Slavin, R. E. (2013). The effectiveness of educational technology applications for enhancing mathematics achievement in K-12 classrooms: A meta-analysis. *Educational Research Review, 9*, 88–113.

Dreyfus, T. (2020). Abstraction in mathematics education. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 13–16).

Drijvers, P. (2015). Digital technology in mathematics education: Why it works (or doesn't). Selected regular lectures from the 12th international congress on mathematical education,

Drijvers, P., Ball, L., Barzel, B., Kathleen Heid, M., Cao, Y., & Maschietto, M. (2016). *Uses of technology in lower secondary mathematics education: A concise topical survey*. Springer Nature.

Forsström, S. E., & Kaufmann, O. T. (2018, December 2018). A literature review exploring the use of programming in mathematics education. *International Journal of Learning, Teaching and Educational Reseach, 17*(12), 18–32.

Greefrath, G., & Siller, H.-S. (2017). Modelling and simulation with the help of digital tools. In *Mathematical modelling and applications* (pp. 529–539). Springer.

Harel, G., & Sowder, L. (2005). Advanced mathematical-thinking at any age: Its nature and its development. *Mathematical thinking and learning, 7*(1), 27–50.

Hayes, J., & Stewart, I. (2016). Comparing the effects of derived relational training and computer coding on intellectual potential in school-age children. *British Journal of Educational Psychology, 86*(3), 397–411.

Heintz, F., Mannila, L., Nordén, L.-Å., Parnes, P., & Regnell, B. (2017). Introducing programming and digital competence in Swedish K-9 education. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives,

Hiebert, J. (1984). Children's mathematics learning: The struggle to link form and understanding. *The elementary school journal, 84*(5), 497-513.

Hiebert, J. (2013). *Conceptual and procedural knowledge: The case of mathematics*. Routledge.

Hillel, J. (1992). The computer as a problem-solving tool: It gets a job done, but is it always appropriate? In *Mathematical problem solving and new information technologies* (pp. 205–218). Springer.

Hoffman, B., & Spatariu, A. (2008). The influence of self-efficacy and metacognitive prompting on math problem-solving efficiency. *Contemporary Educational Psychology, 33*(4), 875–893.
Johnson, H. L., Coles, A., & Clarke, D. (2017). Mathematical tasks and the student: Navigating 'tensions of intentions' between designers, teachers, and students. *Zdm, 49*(6), 813–822.
Joubert, M. V. (2007). *Classroom mathematical learning with computers: The mediational effects of the computer, the teacher and the task.* [Doctoral thesis, University of Bristol].
Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13*(1), 33–50.
Kieran, C. (2019). Task design frameworks in mathematics education research: An example of a domain-specific frame for algebra learning with technological tools. *Compendium for Early Career Researchers in Mathematics Education*, 265.
Kilpatrick, J., Swafford, J., & Findell, B. (2001). The strands of mathematical proficiency. *Adding it up: Helping children learn mathematics*, 115–118.
Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. 2004 IEEE Symposium on Visual Languages-Human Centric Computing,
Krainer, K. (1993). Powerful tasks: A contribution to a high level of acting and reflecting in mathematics instruction. *Educational Studies in Mathematics, 24*(1), 65–93.
Laborde, C., & Sträßer, R. (2010). Place and use of new technology in the teaching of mathematics: ICMI activities in the past 25 years. *Zdm, 42*(1), 121–133.
Leung, A., & Baccaglini-Frank, A. (2016). *Digital Technologies in Designing Mathematics Education Tasks: Potential and Pitfalls* (Vol. 8). Springer.
Leung, A., & Bolite-Frant, J. (2015). Designing mathematics tasks: The role of tools. In *Task design in mathematics education* (pp. 191–225). Springer.
Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: logo vs. scratch. *Proceedings of the 41st ACM technical symposium on computer science education.*,
Li, Q., & Ma, X. (2010). A meta-analysis of the effects of computer technology on school students' mathematics learning. *Educational Psychology Review, 22*(3), 215–243.
Lithner, J. (2017). Principles for designing mathematical tasks that enhance imitative and creative reasoning. *Zdm, 49*(6), 937–949.
Lockwood, E., DeJarnette, A. F., Asay, A., & Thomas, M. (2016). Algorithmic thinking: An initial characterization of computational thinking in mathematics. *North American Chapter of the International Group for the Psychology of Mathematics Education*.
McGonigal, J. (2011). *Reality is broken: Why games make us better and how they can change the world*. Penguin.
Misfeldt, M., & Ejsing-Duun, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. *CERME 9-Ninth Congress of the European Society for Research in Mathematics Education*,
Mitchelmore, M., & White, P. (2007). *Abstraction in mathematics learning* (Vol. 19). Springer.
NCTM. (2014). *Principles to actions : ensuring mathematical success for all*. National Council of Teachers of Mathematics.
Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings: Learning cultures and computers* (Vol. 17). Springer Science & Business Media.
OECD. (2015). *Students, computers and learning: Making the connection.* PISA, OECD Publishing, Paris. https://doi.org/doi:https://doi.org/10.1787/9789264239555-en
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
Piaget, J. (1964). Part I: Cognitive development in children: Piaget development and learning. *Journal of Research in Science Teaching, 2*(3), 176–186.
Polya, G. (1957). *How to solve it: A new aspect of mathematical methods*. Prentice University Press.
Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers and Education, 128*, 365–376.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science, 45*(5), 583–602.
Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1–24.
Rabardel, P. (2002). *People and technology: A cognitive approach to contemporary instruments.* https://hal.archives-ouvertes.fr/hal-01020705
Rakes, C. R., Valentine, J. C., McGatha, M. B., & Ronau, R. N. (2010). Methods of instructional improvement in algebra: A systematic review and meta-analysis. *Review of educational research, 80*(3), 372–400.
Ran, H., Kasli, M., & Secada, W. G. (2020). A meta-analysis on computer technology intervention effects on mathematics achievement for low-performing students in K-12 classrooms. *Journal of Educational Computing Research*, 0735633120952063.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., & Silverman, B. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.
Ross, S., States, U., Morrison, G., Lowther, D. (2010). Educational technology research past and present: balancing rigor and relevance to impact school learning. *Contemporary Education Technology, 1*, 17-35.
Ruthven, K., Laborde, C., Leach, J., & Tiberghien, A. (2009). Design tools in didactical research: Instrumenting the epistemological and cognitive aspects of the design of teaching sequences. *Educational Researcher, 38*(5), 329–342.
Schoenfeld, A. H. (1985). *Mathematical problem solving*. Academic Press, New York, NY.
Schoenfeld, A. H. (2013). Reflections on problem solving theory and practice. *The Mathematics Enthusiast, 10*(1), 9–34.
Sierpinska, A. (2004). Research in mathematics education through a keyhole: Task problematization. *For the learning of mathematics, 24*(2), 7–15.
Sinclair, N., Bussi, M. G. B., de Villiers, M., Jones, K., Kortenkamp, U., Leung, A., & Owens, K. (2016). Recent research on geometry education: An ICME-13 survey team report. *Zdm, 48*(5), 691–719.
Skemp, R. (1986). The psychology of mathematics learning. *Suffolk: Penguin Books*.
Skemp, R. R. (1976). Relational understanding and instrumental understanding. *Mathematics Teaching, 77*(1), 20–26.
Soloway, E. (1993). Should we teach students to program? *Communications of the ACM, 36*(10), 21–24.
Star, J. R. (2005). Reconceptualizing procedural knowledge. *Journal for Research in Mathematics Education*, 404–411.
Stein, M. K., Grover, B. W., & Henningsen, M. (1996). Building student capacity for mathematical thinking and reasoning: An analysis of mathematical tasks used in reform classrooms. *American Educational Research Journal, 33*(2), 455–488.
Stein, M. K., & Smith, M. S. (1998). Mathematical tasks as a framework for reflection: From research to practice. *Mathematics Teaching in the Middle School, 3*(4), 268–275.
Stephens, M. (2018). Embedding algorithmic thinking more clearly in the mathematics curriculum. *Proceedings of ICMI Study, 24*, 483–490.
Stephens, M., & Kadijevich, D. M. (2020). Computational/algorithmic thinking. In *Encyclopedia of mathematics education* (pp. 117–123).
Stillman, G. (2004). Strategies employed by upper secondary students for overcoming or exploiting conditions affecting accessibility of applications tasks. *Mathematics Education Research Journal, 16*(1), 41–71.

Sullivan, P., Clarke, D., & Clarke, B. (2012). *Teaching with tasks for effective mathematics learning* (Vol. 9). Springer Science & Business Media.

Szlávi, P., & Zsakó, L. (2006). Programming versus application. International Conference on Informatics in Secondary Schools-Evolution and Perspectives,

Watson, A., Ohtani, M., & Ainley, J. (2015). Task design in mathematics education. *Proceedings of ICMI Study, 22*.

Wiliam, D., & Thompson, M. (2008). Integrating assessment with learning: What will it take to make it work? In *The future of assessment: Shaping teaching and learning* (pp. 53–82). Routledge.

Wittmann, E. C. (1995). Mathematics education as a 'design science'. *Educational Studies in Mathematics, 29*(4), 355–374.

Woo, Y., & Reeves, T. (2006). *Meaningful online learning: Exploring interaction in a web-based learning environment using authentic tasks* [Doctoral thesis, University of Georgia].

Yackel, E. (2002). What we can learn from analyzing the teacher's role in collective argumentation. *The Journal of Mathematical Behavior, 21*(4), 423–440.

Young, J. (2017). Technology-enhanced mathematics instruction: A second-order meta-analysis of 30 years of research. *Educational Research Review, 22*, 19–33.

Zichermann, G., & Linder, J. (2010). *Game-based marketing: inspire customer loyalty through rewards, challenges, and contests*. John Wiley & Sons.

Facilitating exploratory talk through mathematical programming problems

Abstract

With several Nordic countries implementing programming into their curricula, there is a need for research into the combination of mathematical learning and programming. With tasks being the main "thing to do" in the mathematics classroom and exploratory talk being closely linked to learning this article investigates what contributes to and what hinders exploratory talk when working on mathematical programming problems. The data collection comprises video and audio recordings of students working on tasks in a mathematics classroom in Norway. The findings include a set of recommendations for implementation and design of mathematical programming tasks into the mathematics classroom.

1 Introduction

In the last few years, programming has received focus primarily due to its implementation into the national mathematics curricula of several countries (Sentance & Csizmadia, 2015). The combination of countries including programming in their mathematics curricula (Bocconi et al., 2018) and the lack of research into how to facilitate the implementation of such programming (Weintrop et al., 2016) is a challenge. Previous research has shown that programming improves students' logical thinking (Park et al., 2015) and their understanding of mathematical processes (Calao et al., 2015), which can lead to more joyful learning processes (Djurdjevic-Pahl et al., 2016) and strengthen students' self-confidence (Shim et al., 2016). Combining student interaction with task design, where students rely on each other to generate, challenge, refine and pursue new ideas, has been shown to be beneficial (Francisco & Maher, 2005). Learning occur when small groups of students work on mathematical problems as such tasks facilitate the students' interactions and help them construct new ideas and discover new ways of thinking when applying mathematical understanding (Martin et al., 2006). Small group collaborative learning in school mathematics, if conducted appropriately, can bring about more equal academic success amongst all students compared to traditional methods of teaching (Davidson & Kroll, 1991; DePree, 1998; Slavin, 1990; Urion & Davidson, 1992). Exploratory talk, where students engage critically but constructively with each other in small groups (Mercer, 2005; Mercer & Littleton, 2007), is used here as it has been shown to stimulate subject learning and reasoning skills (Knight & Mercer, 2015; Mercer et al., 2004; Mercer & Sams, 2006).

A mathematical programming problem (MPP) is a series of tasks designed for students to combine mathematics with programming to resolve a problem (Munthe, 2022, submitted). The article presents the design of MPPs and their focus on facilitating exploratory talk, before explaining the design of the MPPs used here. Through an analysis of the interaction between students working in groups on mathematical programming problems, the article investigates elements contributing to and hindering exploratory talk.

2 The design of MPPs

A review of research into computer education indicates that learning to program is difficult as "students exhibit various misconceptions and other difficulties in syntactic knowledge, conceptual knowledge, and strategic knowledge" (Qian & Lehman, 2017, p. 17). Ko et al. (2004) use the term *barriers* to differentiate between six different types of adversities students encounter when learning

to build a program. These barriers are a combination of syntax errors, structural errors, logical challenges, error handling issues and problems related to the number of commands available (Ko et al., 2004). When combined with the difficulty of creating tasks using digital technologies (Joubert, 2007; Laborde & Sträßer, 2010), programming is a complex tool to be introduced and implemented into mathematics classrooms.

Mathematical learning is more likely to take place when students are committed to a solving a problem, and Brousseau (1997, p. 83) defines, in the theory of didactical situations (TDS), knowledge as a property of a system consisting of a subject and a milieu. Designing problems that facilitate students adapting their strategies to obtain the desired knowledge is challenging, and MPPs accomplish this through the design of problems that students can overcome while avoiding many adversities. As students work on the problems, they interact with the milieu, which can be both collaborative and antagonistic. This period of interaction is the *adidactical situation*, where students show initiative and responsibility for the outcome of the learning process. In the TDS, an epistemological obstacle constitutes a form of knowledge that has been relevant and successful in particular contexts, often school contexts, but that becomes false or insufficient at a particular moment in time. If students overcome such an obstacle through the adaptation of their strategies, the desired knowledge can be obtained, thereby generating an epistemological obstacle. Ontogenic obstacles relate to the limitations of students and a lack of required prior learning, and didactical obstacles relate to the presentation of the subject, "the result of narrow or faulty instruction" (Harel & Sowder, 2005, p. 34). If students encounter and overcome obstacles as they are working through the given problems, adaptation may take place. Similarly, Stein et al. (1996, p. 426) state that "tasks used in mathematics classrooms highly influence the kinds of thinking processes in which students engage, which, in turn, influences student learning outcomes." The design of MPPs is focused on creating adidactical situations (Brousseau, 1997), facilitating exploratory talk during which students reconsider their strategies, develop new pathways, discuss with their peers, conjecture and experiment, all related to the intended learning process (Leung & Baccaglini-Frank, 2016). To facilitate adidactical situations (Brousseau, 1997), an MPP is structured using seven steps, presented in Figure 1.

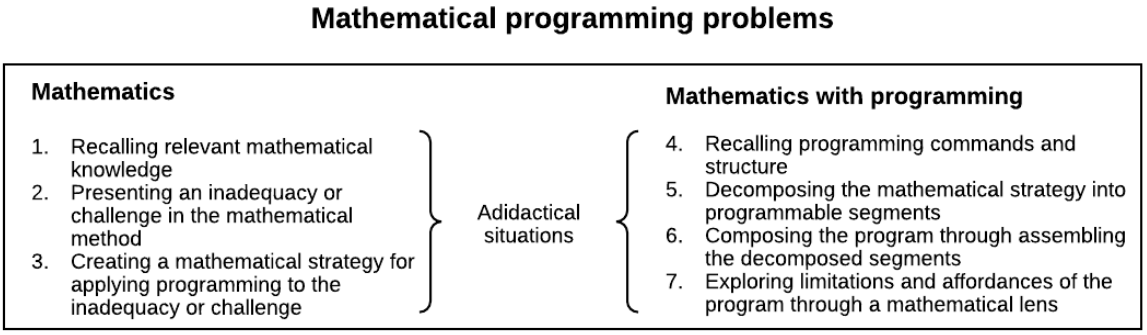


Figure 1: The seven steps for designing MPPs.

The structure of the design is the product of a three-year iterative process of implementing MPPs in an upper-secondary school mathematics class (Munthe, 2022, submitted). As an example, I will use the building of a program to calculate the zero-point of a function using the bisectional method[1].

[1] The bisectional method is a root-finding method applicable to any continuous function for which two values with opposite signs are known. The method consists of repeatedly bisecting the interval defined by the two values and then selecting the subinterval in which the function changes sign and that therefore contains the root (intermediate value theorem).

Step 1 consists of asking the students to find the zero-points for several different types of functions, illustrating the different methods required for each type. Step 2 problematizes finding the zero-point if the given function, such as $f(x) = \ln(x + 2) - x$, does not conform to any set method. In step 3, one student takes the role of a "program" and is given a set of graphs where the function is unknown, but the zero-points are provided. The rest of the group is tasked with finding the zero-points by giving the "program" x-coordinates, and the "program" can only respond with the accompanying function values. This allows several strategies for finding the zero-point to be tested and evaluated. In step 4, tasks allowing students to recollect necessary programming structures and commands are given. Step 5 consists of transforming the method discovered in step 3 into a coding structure, and step 6 consists of building the program. Both steps 5 and 6 contain tasks to assist with aspects of code building. The final step asks the students to evaluate the program, investigate whether it works for all functions and provide explanations of why it does not work for some functions if that is the case.

3 Theory

Learning in mathematics can be seen as "the construction of a web of connections – between classes of problems, mathematical objects and relationships, real entities and personal situation-specific experiences" (Noss & Hoyles, 1996, p. 105). The linking of concepts is also well established in the adidactical situations of Brousseau (1997). Facilitating students in explaining their reasoning and providing warrants for their arguments while recognizing them as contributors are part of the exploratory talk that takes place between the students (Choi & Walters, 2018; Mercer, 2005). Exploratory talk is when "partners engage critically but constructively with each other's ideas. Statements and suggestions are offered for joint consideration. These may be challenged and counter-challenged, but challenges are justified, and alternative hypotheses are offered. Partners all actively participate, and opinions are sought and considered before decisions are jointly made" (Mercer, 2005, p. 9). Such exploratory talk increases students' conceptual understanding their ability to reason in terms of mathematical problem-solving (Lampert et al., 1996; Lehrer & Schauble, 2005; Michaels et al., 2008; Resnick et al., 1992; Yackel & Cobb, 1996). The role of mathematical talk and discussion in the effective teaching of mathematics is cemented in research (e.g. Cobb et al., 1997; Kazemi & Stipek, 2009; Nathan & Knuth, 2003; Resnick et al., 2017; Sfard, 2000), and the foundation for this article is that exploratory talk promotes mathematical learning.

Learning mathematics in small groups may also be beneficial when working with computers (Berry & Sahlberg, 2006), and Lou et al. (2001, p. 449) state that when combining learning with technology and exploration, "small group learning had significantly more positive effects than individual learning on student individual achievement." The exploratory talk students undertake in groups consists of several steps. Starting with an initializer which can be a claim, a suggestion, or simply the reading of the problem. A claim is defined by Toulmin (1969) as an assertion to be taken seriously, separating it from more hypothetical and frivolous statements. A suggestion can be in the form of stating a hypothesis (Pedaste et al., 2015) and will have different claims to attention. A claim and a suggestion are similar in that they are both put forward, but they differ in the confidence of their statements. When the students are faced with a problem, it often requires them to recall previous knowledge to initiate the talk, either by inspiring others to utilize it or by expressing a perspective through a "what if?" type of question (Alrö & Skovsmose, 2004). This collective reflection (Tabach & Schwarz, 2018; Yackel & Cobb, 1996) creates the foundation for both solving the problem and the creation of new knowledge. This is not limited to the initial part of the discussion and can move the dialogue in another direction (Alrö & Skovsmose, 2004). Gellert (2014) states that the use of initiation is where

one or more perspectives of the problem are developed and built together with a need for clarification.

The usage of short exercises promoting curiosity and eliciting prior knowledge is essential to facilitate engagement, and the exploration that follows is the result of a combination of promoting curiosity and accessing previous knowledge to ensure that the students are prepared for the learning outcomes of the current problem (Bybee et al., 2006). Curiosity, when integrated with presenting a challenge, can facilitate the process of engagement (Pedaste et al., 2015). Exploration is a sequence of activities initiated by the problem and undertaken by the students, using prior knowledge to generate new ideas, explore a range of possibilities within the problem and undertake an investigation (Bybee et al., 2006). Both engagement and exploration are observable through the exploratory talk the students use to engage critically, but constructively, with each other's ideas (Mercer, 2005).

Explanation involves the students demonstrating their conceptual understanding of the engagement and exploration undertaken (Bybee et al., 2006). Students' progression towards the learning objectives of a task is facilitated by them explaining their understanding of a concept or result to each other (Bybee et al., 2006). Exploratory talk also consists of explanation and evaluation, where making meaning out of the collected data to synthesize new knowledge (Pedaste et al., 2015) and challenging or counterchallenging a proposal are central. This also includes making the reasoning of the group visible (Mercer, 2005), which involves each individual stating what they think to the group and being open to examination from others (Alrö & Skovsmose, 2004). The process of engaging in these reflective activities facilitates learning (Pedaste et al., 2015). The result of the explanation and evaluation parts of exploratory talk can move mathematical talk in a new direction (Alrö & Skovsmose, 2004; Gellert, 2014). Explanation and evaluation can result in a joint decision (Mercer, 2005) or agreement regarding the clarification of a given problem or a specific result. Such agreement needs to be justified, and a deeper explanation is required, including a detailed account of how the group reached its conclusion (Drageset, 2014). Agreement is like closing (Gellert, 2014), which is ending the current discussion about a given problem. The validity of the agreement depends on what Toulmin (1969) refers to as the modal qualifier, in which the degree of confidence in the conclusion qualifies it as a solution.

Tasks that facilitate mathematical learning using programming also need to avoid overloading students with complexity and programming syntax (Ko et al., 2004). Research has shown that complex problem-solving without sufficient support structures can result in an unproductive cognitive process (Kirschner et al., 2006; Reiser, 2004). In task design, insufficient support structures often result in either didactical or ontogenic obstacles (Brousseau, 1997). Based on this research, this article collectively refers to *adversity* when the students encounter an obstacle, whether it be epistemological, ontogenic or didactical.

4 Method

The MPPs investigated were implemented in a classroom as part of an advanced mathematics course for students (N = 28) aged 17 in the second to last year of secondary school. Three groups were chosen based on a combination of different genders and grades, and most importantly, on the individual students' ability to sufficiently convey their ideas vocally during a lesson. One of the groups consisted of four males, the second one was composed of two females and the third one was made up of one male and one female, all of whom had collaborated well throughout the year. The students were told to discuss the problems within their groups, allowing them to work on and discuss any adversities before receiving help from the teacher.

The data collection consisted of recording each student's computer screen together with their voice while working on the problems. The screen and voice recordings for each member of the group were digitally combined to form one video file with two to four screens and an audio where all of the students' voice recordings were combined. This resulted in several advantages with regard to the transcription. First, viewing all the screens together provided a clear indication of where each student was in the process of building the program. Second, the combination of voice recordings made each student's contribution clear, allowing for an accurate transcription.

The analytical framework used consists of three layers of coding. The *segment* category divides the lesson into smaller sections, in which the students talk about one problem or challenge encountered. A segment is defined as *one problem or one part of a problem, initialized by a problem separate from the previous one, and ending with either an agreement regarding the given problem or the change to a new problem*.

The *subject* category refers to the topic of the interaction. The first type is mathematical, in which the students talk about a mathematical object, such as the derivative or the number of solutions. The second is programming, in which the students specifically talk about programming, for instance, how different commands work, the structure of a code segment or what caused a programming error. The final group is a combination of the two, including talks about how to combine programming language to complete a mathematical objective, for instance, "How do we avoid the program dividing by zero?" or "I want the program to do <a mathematical procedure>. How do I accomplish that?"

The *interaction* category consists of exploratory talk and adversity. Exploratory talk consists of initiation, explanation, exploration and agreement. Initiation is a claim, a suggestion, a recalling of previous knowledge or simply the reading of the problem. Initiation allows for the compartmentalization of the transcript into segments, where each segment starts with one or more of the above statements. Explanation and evaluation are part of the exploratory talk between students, where they explain and argue for their solutions, present criticisms of a solution or suggestion and perform checks to validate their solutions (Mercer, 2005). Evaluation is a natural part of mathematical programming since when the students encounter errors, they need to understand how to resolve them through conversations with their group members (Benton et al., 2017). Mathematical evaluation is valuable as it combines the workings of the program with students' mathematical know-how to build their knowledge networks (Noss & Hoyles, 1996), allowing the students to experience an adidactical situation (Brousseau, 1997). Explanation and evaluation also contain the recollection of previous knowledge as this is required to explain mathematical and combination challenges (Pirie & Martin, 2000). The design facilitates the students' recollection of previous knowledge, which facilitates the building of the program (Pirie & Martin, 2000). Exploration involves the students applying mathematics and/or programming to investigate and explore different approaches to a problem or program (Benton et al., 2017). Such exploration can involve testing for different code snippets and the simple colouring of graphs to the more complex insertion of if-statements, preventing the program from calculating the square root of negative values. The testing can also be mathematical, in which the students explore the program's ability to display the derivative of a complex function, followed by an evaluation of the result. Through exploration, the students investigate the limits and possibilities of the program, leading to mathematical questions, such as "why is the derivative of $\log(x)$ not showing for negative values?" and the mathematical evaluation of the results, such as "why is the result of the calculation not as expected?" Exploration can cause a change in knowledge when the results deviate from the expected, and an evaluation of the mathematics involved is required, essentially creating an epistemological obstacle. Exploration,

together with the adidactical situation, is therefore closely linked to explanation. Agreement is the students collectively reaching a common ground whether it is regarding a specific answer or a code snippet. If a statement is disputed, it is not recognized as an agreement. An agreement can be purely mathematical, code related or a combination of the two. Agreement can mark the end of a problem or can initiate a new proposal, evaluation or exploration. Not every segment reaches an agreement immediately as sometimes explanation and exploration drive the students towards other initiations. If the students are unable to reach an agreement through their own interaction, they usually result to asking the teacher, who, through discussion with the students, assists them in reaching an agreement.

Adversity is defined here as situations where the students display uncertainty regarding how to proceed with the MPP they are working on. From a mathematical perspective, these are the instances where the students encounter obstacles (Brousseau, 1997), and from a programming viewpoint, barriers (Ko et al., 2004). These can be observed as frustration, and commonly occur when executing the program and receiving an error message, but also appear in terms of difficulties distinguishing between commands, sequencing challenges, uncertainty regarding how to proceed and a general sense of a negative premonition as in "this is not going to work." This feedback can lead to discouraging self-relevant interpretations (Fyfe & Brown, 2020), which make the students give up and link their frustrations to mathematics. Adversity by itself does not distinguish between different "types" of obstacles. Only when viewing adversities in relation to exploratory talk does the difference between an epistemological and another type of obstacle become clear. The expectation is that a "good" MPP displays more instances of exploratory talk, whereas a "less good" MPP displays more instances of adversity.

Within each segment, the transcription was coded for both subject and interaction, enabling an analysis of the relationship between exploratory talk and adversity and their relation to mathematics, programming or a combination of the two. As an example, when a student engages in a *mathematical explanation,* they use their mathematical knowledge and vocabulary to present their reasoning. In a *combination explanation,* they include either the code itself or the output from the program to assist their reasoning, and in a *programming explanation*, they use the different commands and programming structure for their reasoning and do not apply mathematical vocabulary.

Table 1: Coding scheme for the analysis of the transcripts.

Code for interaction	Description
Exploratory talk	Engagement within the group consisting of ideas, suggestions, challenges and justifications
Initiation	Start of a segment
Explanation	Explaining and arguing for a solution
	Presenting criticism and suggestions
	Validating a solution
Exploration	Testing a code
	Running the program
Agreement	Reaching common ground
Adversity	The group displays uncertainty regarding how to proceed
Positive	Leading to or facilitating exploratory talk (epistemological obstacles)
Negative	Leading to frustration and a "this is not going to

work”-mentality (didactical and/or ontogenic obstacles)

The first MPP concerns the bisectional method (see the example described above in the design of the MPP). To illustrate a problem containing an adversity that intends to facilitate exploratory talk from the MPP concerning the bisectional method, see Figure 2.

Imagine a machine that contains and unknown linear function and will give you a function value when you input a numerical x-value. All non-numerical inputs will yield an error.

How can you systematically use this machine to find the zero-point for the unknown function to the right? What if the function is non-linear?

The machine can be as accurate as you would like concerning the function value. Discuss possible strategies with your group.

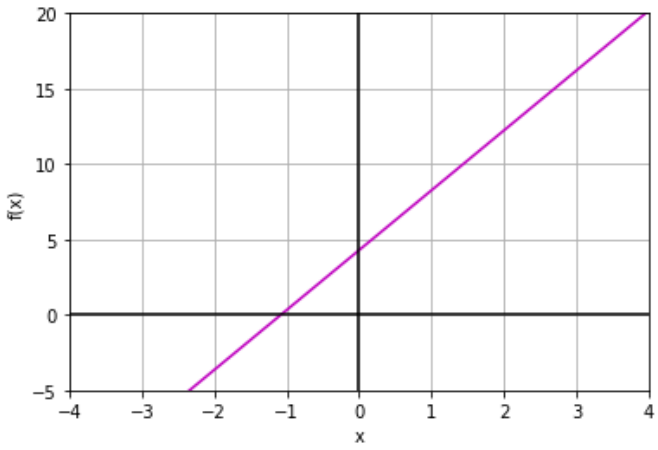


Figure 2: One problem from the first MPP covering the bisectional method.

The second MPP asks the students to build a program implementing Newton’s method for approximating the zero-point[2]. The MPP starts with a recollection of the derivative and its application within function analysis. The MPP facilitates the students in building visual and algorithmic proof for the method, followed by implementing the method into a program. When the students have built the mathematical argument for Newton’s method, they are asked to evaluate it (see Figures 2 and 3).

Does Newton’s method work for all functions?

a) Draw a function that crosses the x-axis in a coordinate system. Choose an x-value close to the zero-point and draw a tangent line to find a new value for the zero-point. Continue this process a few times. Do you get any closer to the zero-point with each new tangent line?

b) Repeat a) a couple of times with different types of functions. Are there certain functions or types of zero-points for which the method does not work? If so, why? Discuss with your group.

Figure 2: One problem from the second MPP covering Newton’s method.

When the students have finished building their programs, they are asked to complete another similar problem (see Figure 3).

[2] Also known as the Newton-Raphson method. It is a root-finding method with an initial guess (x_n) followed by the finding of the tangent line of the function at this point. The next estimate is where the tangent line crosses the x-axis. Algebraically, the estimate calculated can be expressed as $x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$.

Does Newton's method work for all functions?

a) Test the program for several different functions, e.g. different polynomial functions, exponential functions, logarithmic functions.

b) What characterizes the functions for which the program does not work? Discuss with your group.

Figure 3: One problem from the second MPP covering Newton's method.

Results

4.1 Bisectional method

Prior to this lesson, the students completed programming tasks involving building programs covering graph plotting, calculating areas and circumferences of geometric shapes, and solving linear and quadratic equations. This MPP consisted of the students building a program to find the zero-point for a function that crosses the x-axis using the bisectional method. The students were familiar with different methods for finding the zero-point of functions, but had not previously used a numerical method to find the zero-point.

The following extract illustrates the development of a strategy by asking one student from each group to simulate the machine (see step 3 of the example described above in the design of the MPP). The machine-simulation student is A and the zero-searching students are B and C.

Table 2: Transcript of students working on the MPP covering the bisectional method.

	Student	Transcript	Code
101	A	What am I supposed to do now?	Combination – Adversity
102	B	You are now python [the programming language]	Combination – Explanation
103	C	You are going to say …	
104	B	For example, what is y when x is …?	
<short pause while A looks at the graphs given>			
105	B	What is y when x equals zero?	Combination – Exploration
106	A	Then it [the y-value] is minus one	
107	B	But then we take x equals one, right?	
108	A	x equals one … that is ehm … yes, that is approximately, approximately zero point eight	
109	B	ahh, but then we have at least got a zero-point in between there	Mathematics – Explanation
110	A and C	that is so true	Mathematics – Agreement
111	C	We can actually make it even more accurate and say that since we know there is a zero-point between these numbers …	Combination – Explanation

The group have read the problem, and A has just agreed to simulate the machine. The transcript starts with student A expressing initial uncertainty about what to do (101). This is quickly resolved by the other members of the group through explanation and an example (102–104). After a short pause, where student A studies the graphs, the students start enquiring about the function values

for a couple of x-values (105–108). Drawing on the information gathered, a mathematical evaluation and suggestion is made (109) and swiftly agreed upon by the rest of the group (110). This agreement also encompasses verification since the machine-simulating student confirms the suggestion. Finally, student C develops the idea further with both a suggestion and an explanation for the proposal. As indicated by the code column, the segment contains combination adversity and combination and mathematics exploratory talk.

The MPP elicited both success and frustration: success in terms of progress towards building a better method for guessing, and frustration in that the function could have more than one zero-point, both contributing to an adidactical situation. There were some adversities in the implementation of the method they uncovered in the first part, but they were mostly able to build the program. There were several instances where the students explored the program by inputting different types of functions. The interaction between the students consisted of discussing the code and the mathematical verification of the result based on the output of the program. The overall impression obtained from the lesson was that the students understood the bisectional method.

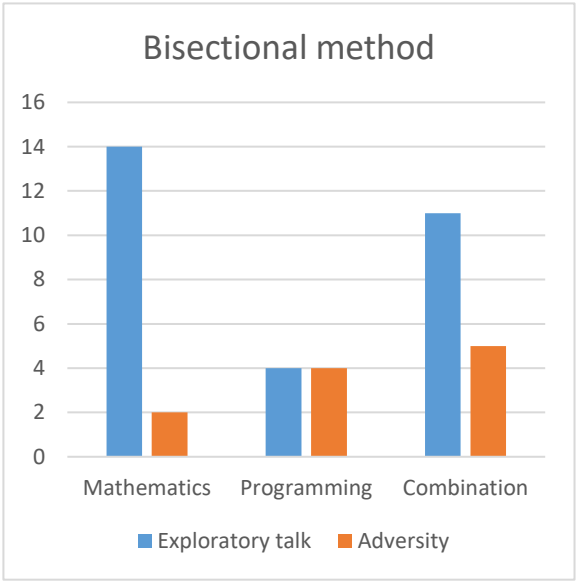


Figure 4: Graph showing instances of the categories in the transcript.

The students' previous knowledge of the derivative and finding zero-points is visualized in the exploratory talk portion of the graph, with almost every single transcript segment relating to mathematics. Since the program did not demand any overly complicated code, the students were observed as having fewer problems with the coding part of the MPP, hence the low incidents of programming dialogue. Additionally, since the MPP was focused on revisiting previous knowledge and games at the start, there were few instances of adversity. Each exploratory talk, especially regarding mathematics and combination, involved several steps and outnumbered the incidents of adversities in the lesson. A typical occurrence to check the validity of the program involved finding a suitable function and solving it mathematically by hand, followed by inputting the function and an initial guess into the program. When the program produced a result, they compared the output to the exact answer and evaluated whether the numerical method was close enough. The few instances of adversity consisted of designed tasks within the MPP, such as, "what is a quick way to check whether numbers have the same or different signs?"; this effectively created adversity in the design to be solved through exploratory talk.

4.2 Newton's method

In between the two classes, the students completed a lesson covering the numerical derivative. The second MPP introduced the students to the combination of both learning a new formula (Newton's method) and implementing the formula into a program. Additionally, the students had never seen a formula that included both a function and its derivative. Combining all these elements is difficult, but the students had now used programming throughout the school year and were expected to be more confident as a result. The MPP started by explaining the origin of Newton's method and why it, in most cases, will yield a good x-value for the zero-point after only a few iterations.

In this transcript, the students are working on a code that calls on a function and have just executed the program and received an error.

Table 3: Transcript of students facing an adversity when working on an MPP.

	Student	Transcript	Code
201	L	but yes, I have something wrong here	Programming – Adversity
202	M	return d y <reading the code>	
203	L	I do not understand what is wrong, but there is something wrong. It says invalid syntax	
204	M	Oh, then you have a wrong parenthesis	Programming – Explanation
205	L	… in this one?	
206	M	In that one <points to the code on the screen>	
207	L	Oh yes, it is missing a bracket here … there <moves the cursor to point at it and inserts a bracket>	Combination – Explanation
<runs the program>			
208	L	Now it works	Combination – Agreement
209	M	Yeah	

After receiving a syntax error, L expresses frustration and cannot initially recognize why the program is at fault (201–203). M recognizes that a syntax error often occurs from a misuse of parentheses (204) before they together locate the line of code containing the error (205–206). By applying an evaluation of both the code and the mathematics, a correction is then made (207) before they run the program again. After the program executes and displays the graph, they agree on the correction made (208–209). The extract contains programming adversity and programming and combination exploratory talk.

The students' conversations were dominated by adversity relating to how to construct the program rather than exploratory talk of how the formula worked. At the end of the lesson, the impression was that the students had constructed a program but had limited knowledge of how the mathematics behind the code worked.

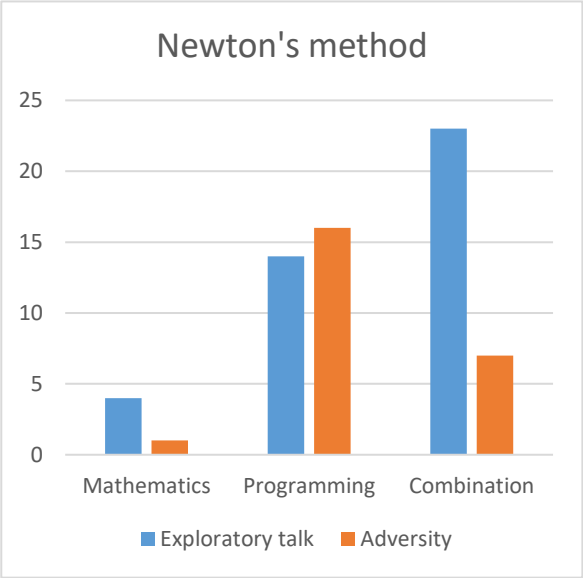


Figure 5: Graph showing instances of the categories in the transcript.

The graph illustrates the prominence of both adversity and exploratory talk concerning programming and the combination of programming and mathematics. It is important to highlight that the combination dialogue, where there is mathematical talk, consisted primarily of the implementation of the formula into the code rather than actual discussion of Newton's formula. The MPP contained exploratory problems, which in the previous lesson had aided the students in discussing the mathematical formula applied, but in this lesson, the students ended the discussion quickly to instrumentally program Newton's formula. As a result, the adversities started out as combination adversity and deteriorated into that related to programming. Most of the programming adversities resulted from the occurrence of an error, which was followed by the students trying to locate the error, often through combination knowledge, before them either succeeding or failing at correcting the error. There were instances where the students completed several iterations of receiving an error, making a correction, running the program and receiving a new or identical error. Every single segment contained programming in one form or another, and the mathematics, although present in the combination dialogue, was lacking.

5 Discussion

Exploratory talk is consistently represented related to either mathematics or the combination of programming and mathematics, indicating that it is possible to implement programming into mathematics classrooms and facilitate exploratory talk. In both of the lessons presented, exploratory talk is present in almost every communication between students, and each lesson reveals a significant number of interactions between the students, which are closely related to learning (Bybee et al., 2006; Mercer, 2005).

The amount of exploratory talk versus adversity is different in the two lessons. The bisectional method lesson contains several instances of adversity, but with the significant amount of exploratory talk that takes place, the adversities are resolved and become an epistemological obstacle {Brousseau, 1997 #177}. The low representation of programming issues is likely due to previous experience with programming, familiarity with the mathematical foundation of the program and the low complexity of the programming required. This lesson indicates how it is

possible to introduce a new subject using programming when students have a strong foundation. When students are learning to apply programming in mathematics, it is beneficial that they are familiar with the mathematical concepts involved so that they are not met with two complexities at the same time (Ko et al., 2004). Applying programming to a well-known mathematical method alleviates one part of this two-sided challenge. In the Newton's method lesson, adversity features more prominently. The expectation was that there would be some discussion regarding the mathematical properties of Newton's formula and its implementation into the program, but only the latter occurred. The lack of mathematical exploratory talk suggests that the MPP did not manage to create a link between these students' previous knowledge of the derivative, the tangent of the derivative and the zero-point of the function. This lesson indicates that when using programming to introduce a new method in mathematics, care must be taken that the programming does not overshadow the mathematical content of the lesson. The difference is that every mathematical step of the bisectional method was already known to the students. In Newton's method, while they had been taught the derivative and application of the tangent, using the derivative to find a zero-point was new. This created an increased amount of adversity for the students, leading to less mathematical exploratory talk.

The bisectional method lesson facilitated exploratory talk indicating that implementing programming into mathematical classrooms can contribute to mathematical learning (Alrö & Skovsmose, 2004; Bybee et al., 2006; Mercer, 2005; Pedaste et al., 2015). Students are better prepared to learn from building a program when there is less adversity and when more support structures are in place (Drijvers, 2015; Kirschner et al., 2018; Reiser, 2004). This is not to say that situations do not exist where both learning a mathematical concept and build a program can occur, but accomplishing both during one lesson is challenging. Designing problems that apply known mathematical concepts together with programming yields opportunities to facilitate a deeper understanding of the underlying mathematics for the learner.

The design idea was that programming can be a tool for learning mathematics. In the bisectional method MPP, the tasks facilitated the students in uncovering the bisectional method themselves before having to create a program using this method, generating a good support system (Drijvers, 2015), contributing to building an adidactical situation (Brousseau, 1997) and facilitating exploratory talk (Mercer, 2005). The lesson contained numerous mathematical exploratory talks, probably due to the introduction and application of well-known methods, together with the fact that the program did not contain any procedures that the students were unfamiliar with. This allowed the students to focus on understanding the mathematics behind the method instead of meandering through programming code, in which didactical and ontological obstacles (Brousseau, 1997) are more likely to present themselves. In the Newton's method MPP, the balance was skewed towards programming, and the exploration suffered because the students focused extensively on getting the program to work, rather than understanding the mathematical principles behind the method. The investigation shows that the creation of a program together with a relatively new mathematical method created ontogenic and didactical obstacles for the students, resulting in them focusing more on the instrumental creation of the program.

The primary adversity hindering exploratory talk that students encounter when working with programming is the building of the program through writing code segments. The hindrance is twofold, where the first obstacle is the coding itself. The implementation of digital tools in the classroom has been researched previously (Drijvers, 2015), but most digital tools used in schools are designed to assist the students using them. They often make use of methods, such as graphical interfaces with buttons that perform certain procedures which are hidden from the user, creating a

"black box" (Buchberger, 1990). Text-based programming is much less lenient and requires the builder to consider both the structure and the commands used to build a functioning program. It is more difficult to build a program that solves a quadratic equation than use a solve command in another digital tool. The combination of understanding the different commands, what they do and how they interact together with understanding the structure of a program is demanding. The second hindrance is the translation of mathematics into a programming code. The mathematical knowledge students possess of how to find the zero-point(s) of a polynomial function is often methodical and different from applying numerical methods. This translation from mathematics to the building of a program requires careful design, where the transition between the two are clear. The two lessons are examples of a "good" and "less good" implementation of programming into a mathematical classroom. A "good" implementation is recognized as containing more mathematical exploratory talk compared to adversity. A successful MPP design uses pre-existing knowledge extensively and then slowly builds on that to facilitate the students' building of the program. Unsuccessful implementation comprises more adversity and less mathematical exploratory talk. Adversities are often related to the syntax and structure of programming, which creates complexity and can hinder mathematical learning (Kirschner et al., 2018; Reiser, 2004). When the program requires mathematics that the students are less familiar with, the complexity entails both programming and mathematics and can potentially create a double hurdle for the students to overcome (Ko et al., 2004).

6 Concluding thoughts

From the discussion above, three main findings can be identified affecting the relevance of MPPs (1), the structure of the design (2, 3) and the choice of the mathematical area represented (2, 3).

1) When implementing programming into mathematics classrooms, it can facilitate mathematical exploratory talk.
2) Programming is best implemented to facilitate the in-depth learning of already-known mathematical concepts as this initiates exploratory talk; however, care is needed when utilizing programming to learn new mathematical concepts.
3) Adversity is both important and challenging when implementing programming into mathematics classrooms. It is important in that it can facilitate mathematical adidactical situations but challenging in that programming adds another layer of complexity.

The data show that programming can facilitate mathematical learning (Kazemi & Stipek, 2009; Lampert et al., 1996; Lehrer & Schauble, 2005; Michaels et al., 2008; Nathan & Knuth, 2003; Resnick et al., 1992; Sfard, 2000; Yackel & Cobb, 1996). An important caveat here is that this article only scratches the surface of adversities. This work was carried out on a small scale, and research with a larger number of students is needed. Further studies focusing more explicitly on the mathematical learning gains from applying programming and how teachers can utilize programming to facilitate learning are necessary.

7 References

Alrö, H., & Skovsmose, O. (2004). Dialogic learning in collaborative investigation. *Nordisk Matematikkdidaktikk*, *9*(2), 39–62.

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, *3*(2), 115–138.

Berry, J., & Sahlberg, P. (2006). Accountability affects the use of small group learning in school mathematics. *Nordic Studies in Mathematics Education*, *11*(1), 5-31.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). The Nordic approach to introducing Computational Thinking and programming in compulsory education. *Report prepared for the Nordic@ BETT2018 Steering Group*, 397-400.

Brousseau, G. (1997). *Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990* (B. N. Cooper M, Sutherland R, Warfield V, Trans.). Kluwer Academic Publishers.

Buchberger, B. (1990). Should students learn integration rules? *ACM Sigsam Bulletin*, *24*(1), 10–17.

Bybee, R. W., Taylor, J. A., Gardner, A., Van Scotter, P., Powell, J. C., Westbrook, A., & Landes, N. (2006). The BSCS 5E instructional model: Origins and effectiveness. *Colorado Springs, Co: BSCS*, *5*, 88-98.

Calao, L. A., Moreno-León, J., Correa, H. E., & Robles, G. (2015). Developing mathematical thinking with scratch. In *Design for teaching and learning in a networked world* (pp. 17-27). Springer.

Choi, J., & Walters, A. (2018). Exploring the Impact of Small-Group Synchronous Discourse Sessions in Online Math Learning. *Online Learning*, *22*(4), 47-64.

Cobb, P., Boufi, A., McClain, K., & Whitenack, J. (1997). Reflective discourse and collective reflection. *Journal for Research in Mathematics Education*, 258-277.

Davidson, N., & Kroll, D. L. (1991). An overview of research on cooperative learning related to mathematics. *Journal for Research in Mathematics Education*, *22*(5), 362-365.

DePree, J. (1998). Small-group instruction: Impact on basic algebra students. *Journal of Developmental Education*, *22*(1), 2.

Djurdjevic-Pahl, A., Pahl, C., Fronza, I., & El Ioini, N. (2016). A pathway into computational thinking in primary schools. International Symposium on Emerging Technologies for Education,

Drageset, O. G. (2014). Redirecting, progressing, and focusing actions—a framework for describing how teachers use students' comments to work with mathematics. *Educational Studies in Mathematics*, *85*(2), 281-304.

Drijvers, P. (2015). Digital technology in mathematics education: Why it works (or doesn't). Selected regular lectures from the 12th international congress on mathematical education,

Francisco, J. M., & Maher, C. A. (2005). Conditions for promoting reasoning in problem solving: Insights from a longitudinal study. *The Journal of Mathematical Behavior*, *24*(3-4), 361-372.

Fyfe, E. R., & Brown, S. A. (2020). This is easy, you can do it! Feedback during mathematics problem solving is more beneficial when students expect to succeed. *Instructional Science*, *48*(1), 23-44.

Gellert, A. (2014). Students discussing mathematics in small-group interactions: opportunities for discursive negotiation processes focused on contentious mathematical issues. *Zdm*, *46*(6), 855-869.

Harel, G., & Sowder, L. (2005). Advanced mathematical-thinking at any age: Its nature and its development. *Mathematical thinking and learning*, *7*(1), 27–50.

Joubert, M. V. (2007). *Classroom mathematical learning with computers: The mediational effects of the computer, the teacher and the task.* [Doctoral thesis, University of Bristol].

Kazemi, E., & Stipek, D. (2009). Promoting conceptual thinking in four upper-elementary mathematics classrooms. *Journal of education*, *189*(1-2), 123-137.

Kirschner, P., Sweller, J., & Clark, R. E. (2006). Why unguided learning does not work: An analysis of the failure of discovery learning, problem-based learning, experiential learning and inquiry-based learning. *Educational Psychologist*, *41*(2), 75-86.

Kirschner, P. A., Sweller, J., Kirschner, F., & Zambrano, J. (2018). From cognitive load theory to collaborative cognitive load theory. *International Journal of Computer-Supported Collaborative Learning*, *13*(2), 213–233.
Knight, S., & Mercer, N. (2015). The role of exploratory talk in classroom search engine tasks. *Technology, Pedagogy and Education*, *24*(3), 303-319.
Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. 2004 IEEE Symposium on Visual Languages-Human Centric Computing,
Laborde, C., & Sträßer, R. (2010). Place and use of new technology in the teaching of mathematics: ICMI activities in the past 25 years. *Zdm*, *42*(1), 121–133.
Lampert, M., Rittenhouse, P., & Crumbaugh, C. (1996). Agreeing to disagree: Developing sociable mathematical discourse. *The handbook of education and human development: New models of learning, teaching, and schooling*, *731*, 764.
Lehrer, R., & Schauble, L. (2005). Developing Modeling and Argument in the Elementary Grades.
Leung, A., & Baccaglini-Frank, A. (2016). *Digital Technologies in Designing Mathematics Education Tasks: Potential and Pitfalls* (Vol. 8). Springer.
Lou, Y., Abrami, P. C., & d'Apollonia, S. (2001). Small group and individual learning with technology: A meta-analysis. *Review of educational research*, *71*(3), 449-521.
Martin, L., Towers, J., & Pirie, S. (2006). Collective mathematical understanding as improvisation. *Mathematical thinking and learning*, *8*(2), 149-183.
Mercer, N. (2005). Sociocultural discourse analysis: Analysing classroom talk as a social mode of thinking. *Journal of Applied Linguistics and Professional Practice*, *1*(2), 137-168.
Mercer, N., Dawes, L., Wegerif, R., & Sams, C. (2004). Reasoning as a scientist: Ways of helping children to use language to learn science. *British educational research journal*, *30*(3), 359-377.
Mercer, N., & Littleton, K. (2007). *Dialogue and the development of children's thinking: A sociocultural approach*. Routledge.
Mercer, N., & Sams, C. (2006). Teaching children how to use language to solve maths problems. *Language and Education*, *20*(6), 507-528.
Michaels, S., O'Connor, C., & Resnick, L. B. (2008). Deliberative discourse idealized and realized: Accountable talk in the classroom and in civic life. *Studies in philosophy and education*, *27*(4), 283-297.
Nathan, M. J., & Knuth, E. J. (2003). A study of whole classroom mathematical discourse and teacher change. *Cognition and instruction*, *21*(2), 175-207.
Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings: Learning cultures and computers* (Vol. 17). Springer Science & Business Media.
Park, I., Kim, D., Oh, J., Jang, Y., & Lim, K. (2015). Learning effects of pedagogical robots with programming in elementary school environments in Korea. *Indian Journal of Science and Technology*, *8*(26), 1-5.
Pedaste, M., Mäeots, M., Siiman, L. A., De Jong, T., Van Riesen, S. A., Kamp, E. T., Manoli, C. C., Zacharia, Z. C., & Tsourlidaki, E. (2015). Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational Research Review*, *14*, 47-61.
Pirie, S., & Martin, L. (2000). The role of collecting in the growth of mathematical understanding. *Mathematics Education Research Journal*, *12*(2), 127-146.
Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1–24.
Reiser, B. J. (2004). Scaffolding complex learning: The mechanisms of structuring and problematizing student work. *The journal of the learning sciences*, *13*(3), 273-304.
Resnick, L., Asterhan, C. S., & Clarke, S. (2017). Student discourse for learning. *Handbook of teaching and learning. Wiley: Wiley-Blackwell*.

Resnick, L. B., Bill, V., & Lesgold, S. (1992). Developing thinking abilities in arithmetic class. *Neo-Piagetian theories of cognitive development: Implications and applications for education*, 210-230.

Ryve, A. (2011). Discourse research in mathematics education: A critical evaluation of 108 journal articles. *Journal for Research in Mathematics Education*, *42*(2), 167–199.

Sentance, S., & Csizmadia, A. (2015). Teachers' perspectives on successful strategies for teaching Computing in school. IFIP TCS,

Sfard, A. (2000). On reform movement and the limits of mathematical discourse. *Mathematical thinking and learning*, *2*(3), 157-189.

Shim, J., Kwon, D., & Lee, W. (2016). The effects of a robot game environment on computer programming education for elementary school students. *IEEE Transactions on Education*, *60*(2), 164-172.

Slavin, R. E. (1990). Achievement effects of ability grouping in secondary schools: A best-evidence synthesis. *Review of educational research*, *60*(3), 471-499.

Stein, M. K., Grover, B. W., & Henningsen, M. (1996). Building student capacity for mathematical thinking and reasoning: An analysis of mathematical tasks used in reform classrooms. *American Educational Research Journal*, *33*(2), 455–488.

Tabach, M., & Schwarz, B. B. (2018). Professional development of mathematics teachers toward the facilitation of small-group collaboration. *Educational Studies in Mathematics*, *97*(3), 273-298.

Toulmin, S. E. (1969). *The uses of argument*. Cambridge university press.

Urion, D. K., & Davidson, N. A. (1992). Student achievement in small-group instruction versus teacher-centered instruction in mathematics. *Problems, Resources, and Issues in Mathematics Undergraduate Studies*, *2*(3), 257-264.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127-147.

Yackel, E., & Cobb, P. (1996). Sociomathematical norms, argumentation, and autonomy in mathematics. *Journal for Research in Mathematics Education*, *27*(4), 458-477.

Programming in the mathematics classroom – The types of adversities students encounter

Keywords: mathematical learning, programming, adversity

Abstract

This article investigates the implementation of mathematical programming problems among upper secondary school students and the types of adversity they experience when working on such problems. The adversities are classified and analysed within a framework of four categories. Concept adversity refers to the use and knowledge of different commands and types in the programming language. Syntax adversity concerns the structure of conditions and loops and the logical build of a program. Output adversity occurs after pressing the 'run the program' button and receiving, for instance, syntax errors, unexpected answers, or no output, and coding adversity is encountered when converting a mathematical procedure to programming code. Alongside several excerpts from transcripts, each adversity is discussed in relation to both mathematical learning and how to mitigate undesirable adversities. Concept adversity was observed to have no relation to mathematical learning, while at the same time not impeding such learning significantly. Syntax and output adversity were observed to contribute to mathematical learning when students started exploring the problems but did not contribute to learning when they were unable to resolve the problem. Coding adversity, when resolved, was observed to facilitate exploration and learning.

Introduction

Programming is currently making its entrance into the mathematics classroom in Nordic countries (Bocconi et al., 2018). With the coupling of mathematics and programming comes a range of possibilities and challenges for learning. Possibilities can take the form of numerical methods and simulations in addition to new ways to teach, learn, and investigate mathematics. Adversity, the focus of this article, can take the form of increased difficulty and complexity, leading to frustration and loss of motivation. In this work, *adversity* is a combination of two terms used in the theory: *obstacles*, from the theory of didactical situations (TDS) in mathematics (Brousseau, 1997) and *barriers*, from the programming community (Ko et al., 2004). When referencing an obstacle, it is mathematical; when referencing a barrier, it is related to programming.

Barriers from the programming community are a combination of syntax errors, structural errors, logical challenges, error handling, and the number of commands available (Ko et al., 2004). Obstacles from the mathematics community are extensive and span a wide selection of areas, from learning the number line in primary school to learning to differentiate and integrate at secondary school (Campbell & Epp, 2005; Merenluoto & Lehtinen, 2004; Nelson & Powell, 2018). Obstacles are important in the process of learning (Lodge et al., 2018), and knowing which obstacles lead to confusion and frustration can facilitate learning (D'Mello et

al., 2014). Obstacles leading to confusion are closely connected to learning in that confusion can have beneficial effects, most importantly when it is resolved (D'Mello & Graesser, 2014). A challenge for educational researchers is to determine which adversities exist and which lead to learning processes and outcomes (Lodge et al., 2018). As Niss (1999) stated, if we know the obstacles that block the paths of students learning mathematics, we will gain a better understanding of how mathematical knowledge and ability are generated and activated, leading to better learning.

Research has addressed the barriers encountered when learning programming (e.g., Grover & Pea, 2013; Jenkins, 2002; Ko et al., 2004; Piteira & Costa, 2013) and the types of adversities students encounter when learning programming as part of mathematics (Benton et al., 2017; Benton et al., 2018). The barriers within programming and the obstacles within mathematics are significant, and this article uses an abductive framework to categorise the adversities met by secondary school students utilising programming in mathematics. In this study, mathematical obstacles experienced by students were continuously investigated and, to explore the mathematical programming adversities they encountered, a framework from word problems (Verschaffel et al., 2020a) was used. An adversity is defined as a situation in which the students display uncertainty regarding how to proceed with the mathematical programming problem (MPP) on which they are working.

This article investigates the types of adversities students encounter when using programming as a tool in mathematics. As well as suggesting how to avoid adversities which do not facilitate learning, this work will aid in both teachers' orchestration of mathematics classes using programming and the design of MPPs. The two research questions are 'What types of adversities do upper secondary school students encounter when working with MPPs in the mathematics classroom?' and 'How are the adversities related to the learning of mathematics?' The aim is to increase our knowledge of the types of adversities that block the paths of students learning mathematics, thus facilitating learning. Following the investigation, suggestions are made on how to mitigate undesirable adversities through both task design and classroom orchestration to facilitate learning.

Mathematical programming problems

MPPs consist of a series of tasks in which students work on building a program to solve a mathematical problem. To facilitate adidactical situations (Brousseau, 1997), the MPP is structured using the seven steps presented in Figure 1.

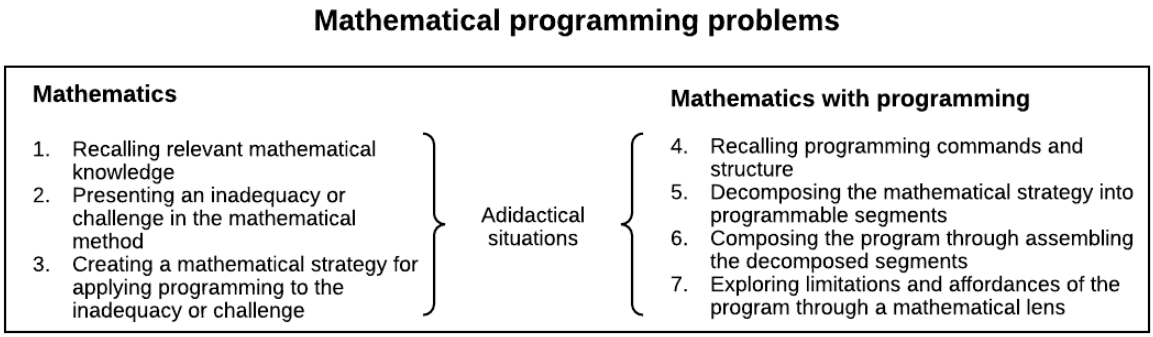


Figure 1: The seven steps for designing MPPs.

The structure of the design is the product of a three-year iterative process implementing MPPs among Norwegian upper secondary school mathematical students. The iterative process consisted of moving from a simple 'build a program that …' to decomposing the MPP into several smaller problems, with questions and activities facilitating discussion and exploration by the students. As a brief example, I use the building of a program calculating the zero point of a function using the bisectional method[1]. Step 1 consists of asking the students to find the zero point of several different types of functions, illustrating the different methods required for each type. Step 2 asks them to find the zero point if the function does not conform to any set method, such as $f(x) = \ln(x + 2) - x$. In step 3, one student takes the role of a 'program' and is given a set of graphs where the function is unknown but the zero points are given. The other members of the group are tasked with finding the zero points by giving the 'program' x-coordinates, and the 'program' can only respond with the corresponding function value. This allows exploration of several strategies for finding the zero point to be evaluated and tested. Step 4 allocates tasks requiring students to recollect necessary programming structures and commands, such as plot commands and while loops. Step 5 consists of transforming the method discovered in step 3 into a coding structure, and step 6 consists of building the program. Both steps 5 and 6 contain tasks to assist in aspects of the building of code, such as the conditions for the while loop. The final step asks the students to evaluate the program, investigate whether it works for all functions, and, if there are functions for which it does not work, determine why. For a more in-depth explanation of the design and examples, see (reference coming, in process). Themes for the MPPs included in this study are the quadratic equation, using the binomial method of finding the zero point, deriving a function numerically, and using Newton's method of finding the zero points[2].

Theory

To discuss the adversities students encounter when working on MPPs, adidactical situations from TDS were applied. An adidactical situation occurs when the students interact with the milieu, consisting of their peers, the tools, and the design of the problems, and take the initiative and have the responsibility for the outcome. In TDS, the obstacles the students encounter are separated into three types (epistemological, didactical, and ontogenic) (Brousseau, 1997), which are used in the discussion of this article. Epistemological obstacles are beneficial and sought after since they force the students to reorganise their knowledge to fit a new situation. Typically, the students are presented with a problem for which their previous knowledge is insufficient or wrong, and this forces them to apply their existing knowledge in a new way. An epistemological obstacle is the desired type of obstacle, facilitating learning as the students overcome the obstacle. Ontogenic obstacles originate from the lack of the knowledge required to make the connection necessary to solve the designed task. Didactical obstacles are caused by erroneous or flawed task design which leaves the students unable to complete the task. Whereas ontogenic obstacles lie more with the students

[1] The bisectional method is a root-finding method applicable to any continuous function for which two values with opposite signs are known. The method consists of repeatedly bisecting the interval defined by the two values and then selecting the subinterval in which the function changes sign, therefore containing the root (intermediate value theorem).

[2] Also known as the Newton–Raphson method, this is a root-finding method in which an initial guess (x_n) is followed by finding the tangent line of the function at this point. The next estimate is where the tangent line crosses the x-axis. Algebraically, the estimate calculated can be expressed as $x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$.

and teacher, didactical obstacles belong firmly to the designer of the task. Obstacles can facilitate learning, such as epistemological obstacles, or hinder or limit it, such as ontogenic and didactical obstacles (Brousseau, 1997). The observed obstacles therefore influence the design, since both ontogenic and didactical obstacles can be mitigated through a design process.

Word problems

The transformation of a mathematical model into a programming code is like the internal construction of a model within word problems, after which it is implemented into a mathematical model. The building and running of a program are like working through a mathematical model to derive a mathematical result. Interpreting and evaluating whether the outcome or output is correct and reasonable is equally important. Finally, communicating the result of the calculation or providing a descriptive output from the program occurs in both instances. Given all these similarities, the implementation of word problem strategies (Verschaffel et al., 2020a; Verschaffel et al., 2000) and adversities was an important starting point for building the framework and performing the analysis of the obstacles encountered by students when solving MPPs.

A word problem is defined as a verbal description of a problem situation where one or a series of questions is posed whose answer can be reached through the use of mathematical operations and numerical data contained within the problem statement (Verschaffel et al., 2000). As with all types of problems, what constitutes a problem for one student may not be a problem for another (Muir et al., 2008). Whether a task constitutes a problem depends on several factors, such as familiarity with the type of problem, ability to recall required knowledge, and the tools available (Verschaffel et al., 2020a; Verschaffel et al., 2020b). The pathway through a word problem is complex, but central aspects of the process are contained within the six phases outlined below (Verschaffel et al., 2020a), which have similarities to the structure (but were not used in the design) of the MPPs (Figure 1).

1) Internal construction of a model depicting the problem, understanding each element and their relation.
2) Transformation of the model into a mathematical model in which the elements and their relation are essential for the solution.
3) Working through the mathematical model to derive a mathematical result.
4) Interpreting the outcome of the computational work.
5) Evaluating whether the outcome is computationally correct and reasonable.
6) Communicating the result.

This is not a sequential model, as there is room to move back and forth and skip phases, depending on student knowledge. Research into word problems indicates that the main obstacles encountered by students are text comprehension, numerical complexity, and the connection between the two (Daroczy et al., 2015; Pongsakdi et al., 2020). These obstacles share properties with both ontogenic and epistemological obstacles (Brousseau, 1997). Given both the known obstacles and the model above, suggestions have been made of how to build a word problem encompassing three main characteristics (Verschaffel et al., 2020a). Firstly, the use of varied, cognitively challenging, and/or realistic tasks lowers the chance that students develop superficial strategies and encourages them to discuss different types of models to apply, facilitating adidactical situations (Brousseau, 1997). Secondly, scaffolding the problem

and providing the students the ability to experiment with and discuss them hinders ontogenic obstacles and builds towards adidactical situations. Thirdly, creating a classroom conducive to the discussion of mathematical problem solving, for instance problem-solving strategies such as those proposed by Polya (1957), further facilitates adidactical situations. Word problems aim to motivate students to study mathematics, facilitate creative thinking, develop their problem-solving abilities, and assist in the development of new mathematical concepts and skills (Verschaffel et al., 2000).

Programming barriers

Challenges in learning programming have been investigated frequently (Medeiros et al., 2018; Piteira & Costa, 2013), and this article focuses on the learning of programming syntax and sequences of variables, loops, and conditions, as these are both the most applicable to a school setting and present a significant challenge to learners (Bosse & Gerosa, 2017). Ko et al. (2004) refer to six barriers in the programming process, three of which are closely related to programming in a school setting. The first is *selection barriers*, which relate to the programming interface and the knowledge of what tools and commands to use to build for a result. The second is *understanding barriers*, which consist of knowledge of how to handle errors and unexpected behaviour of the program. Error handling is a recognised challenge within programming education (Lahtinen et al., 2005). The third is *information barriers*, in which the program did not confirm the result or behave in accordance with the hypotheses, which often results in the inability to evaluate what went wrong. All three of these barriers are related to the evaluation, rather than the execution, of the program. The students' evaluation and discussion of a mathematical problem is central to the adidactical situation. Ko et al. (2004) further divided the barriers into subgroups of surmountable and insurmountable barriers. Surmountable barriers are those the students are able to overcome; when the complexity or number of barriers becomes too large, the students are unable to continue and face an insurmountable barrier. Surmountable barriers bear a resemblance to epistemological obstacles, and insurmountable barriers bear a resemblance to ontological obstacles (Brousseau, 1997).

Most school students are novice programmers, who are limited to surface knowledge of programs, typically using the 'line by line' approach instead of meaningful programming building (Lahtinen et al., 2005). Students often have knowledge of both the syntax and the semantics of programming but lack the ability to combine them into valid and efficient programmes (Winslow, 1996). Despite several attempts at designing strategies to support novice programming (such as collaborative teamwork, peer tutors, workshops, and forums), multiple adversities remain when students are introduced to algorithmic thinking (Stephens & Kadijevich, 2020), logic, and problem solving. The difficulty of algorithmic thinking combined with the exhausting labour of learning syntax generally leads to frustration, rejection, and poor vision (Buitrago Flórez et al., 2017).

Method

Implementation and data collection

MPPs were implemented in a classroom over a period of two years (2018–2020) in an advanced mathematics course for students (N=28) in the last two years of secondary school (age 17–19). The students received some basic knowledge of programming from a 10-hour crash course in Python programming which included variables, input and print, mathematical

operators, conditional statements, and loops. The students participated in a total of 10 lessons during the school year in which programming was used. For each lesson (90 minutes), two groups of two and one group of four students were recorded while working on the task. The grade average of the class was 4.3 (on a scale from 1 to 6, where 6 is the top score), and the class consisted of students spanning the entire grade spectrum.

The students were informed of the implementation prior to their subject election so they could elect not to be a part of this specific class. Initially, there was a concern that this would reduce the validity of the research due to a reduced number of participants and that only either high-achieving students or those with prior knowledge of programming would participate. A review of the students' mathematics grades from the previous year and an anonymous survey asking the students for their knowledge of programming revealed that neither concern was borne out. The three groups consisted of four males, two females, and one male and one female, respectively, all of whom collaborated well throughout the year. The students were told to discuss any adversity within the group before requesting help from the teacher. The data collection consisted of a recording of each student's computer screen together with their voice. Each student in each group had a screencast program installed on their school laptops and was given a lavalier microphone to ensure the best recording of their voice.

Data analysis and analytical framework

The screen and voice recordings for each member of the group were digitally combined to form one video file with two to four screens and an audio where all the students' voice recordings were added together. This facilitated transcription, as viewing all the screens together allowed transcribers to clearly see where each student was in the process of building the program, and the combination of voice recordings made the voice of each student very clear, allowing for an accurate transcription. Every part of the group discussion relating to mathematics was transcribed. Non-mathematical discussion was noted, as it may be important to learning (Ryve, 2011; Wegerif, 2007), but not transcribed. On average, each group had one to two minutes of non-mathematical discussion every 20 minutes. The next step was dividing the transcript into segments where the students encountered an adversity when working on the MPPs. Afterwards, a second reading was performed to categorise what types of adversities were present in each segment and whether they were resolved. This process was an abductive one whereby each segment was read, and the types of adversities were described in detail and then compared with the programming barriers from Ko et al. (2004) and obstacles from TDS (Brousseau, 1997). After an initial framework was created, the transcript was re-read to investigate similarities between segments and their accompanying categories with the aim of uncovering broader categories. As an example, it quickly became apparent that several adversities consisted of the students forgetting either the name or the properties of a command or type, leading to a category called concept adversities. A similar process led to the uncovering of the remaining three categories. After the categories were decided, another reading of the segments was performed to ensure that the categories covered all the adversities in the transcript and that each segment could be coded by concept, syntax, output, or coding.

Concept adversity is related to the use and knowledge of different commands, methods, and types in a programming language and encompasses everything from not recalling a command to not understanding what a command does or a type represents. It corresponds to the selection barriers of Ko et al. (2004) and the ontogenic obstacles of Brousseau (1997). *Syntax*

is closely related to concepts but is a separate category, being more closely related to the logical aspect of programming. Syntax includes the structure of conditions and loops and the logical build of a program. A program calculating the square root of a number first needs to check if the number is positive or zero before calculating; otherwise, there will be an error. Errors in syntax are related to the student's internal construction of a model depicting the problem, understanding each element and the relation between them (Verschaffel et al., 2020a).

Output is when the result after pressing the 'run the program' button presents an adversity, which can take many forms, from syntax errors and unexpected answers to no output at all. Some forms are simple, such as errors concerning a missing bracket indicating an understanding barrier, and some are complex, such as unexpected or no outputs that are hard to evaluate, indicating an information barrier (Ko et al., 2004). Interpreting and evaluating the outcome of the computational work and whether it is correct and reasonable (Verschaffel et al., 2020a) is expected, and the more difficult an output is to evaluate, the more it hinders an adidactical situation. *Coding* is when the students are converting a mathematical procedure to programming code, which is the transformation of the model into a mathematical model (Verschaffel et al., 2020a). The process of transposing a mathematical idea or method to a working programming structure is of particular interest as it facilitates students exploring mathematical methods and concepts with the logical structure of programming to develop an in-depth understanding of mathematics. The exploration also facilitates adidactical situations. Each adversity in our study had an additional code, which distinguished whether the adversity was resolved or not. The framework is presented in Table 1: Framework for investigating the adversities encountered by students when working on MPPs., which shows the subcategories and gives examples from the transcripts.

Table 1: Framework for investigating the adversities encountered by students when working on MPPs.

Type of adversity	Description	Examples from transcript
Concept	Unknown command or type Unable to recall command Unable to recall function of a command	"What is float?" "What does that return mean?"
Syntax	Placement of structure within a program Defining variables Structure of if statements Misunderstanding the sequential reading of code	"Where are we going to input this [line of code]?" "Then we must change the if statement. We need two elif's [else if condition in Python code]. Is that allowed?"
Output	No output Understanding errors Not understanding errors Unexpected answer	"What happened?" "Nothing." "There is something wrong, there is something wrong in line twenty." "Oh yes, should it not be like a … oh, I see what is wrong, there should be a colon there." "Name x is not defined <reads from screen>. Why did that not work?" "What does 'expected indented block' mean?"

		"Why? … Fifty-six, what do you mean by that. It should be three point five … That is … that is very strange."
Coding	Converting known mathematics into code Expanding program	"We need to create the tangent line for the …" "How are we going to write [code for] that?" "We have managed to find the zero points." "But we want to get them all at the same time."

The framework allows the investigation of several types of adversities that affect the interaction between students and their progress through the task. Adversities that are resolved are of special interest since they are closely related to learning (D'Mello et al., 2014), and the framework allows us to distinguish both the type of adversity and how often adversities are resolved. The transcripts presented in the results include extracts from each of the four categories. All the adversities were coded into one or more of the categories. Each excerpt is presented to illustrate a type of adversity and followed by an explanatory description. When a diversity within a category was observed, two or more excerpts are presented to illustrate the similarities and differences between the types of adversities.

Results

The design of the MPP facilitated the students' discussion throughout the lesson, allowing for an extensive investigation. The excerpts illustrate the differences between the types of adversities observed. Throughout the lessons, segments with adversity occurred 51 times in the transcripts. An adversity segment spanned everything from a brief discussion of only a couple of interactions to discussions lasting several minutes within the group. In 11 of the adversity segments, there were occurrences of multiple categories, which explains the sum of occurrences equalling 60.

Table 2 – Numerical result of analysis of transcript

Category	Description	Number of occurrences	Resulted in resolution
Concept [C]	Unable to recall command or type Unable to recall the function of a command	18	14
Syntax [S]	Placement of structure within a program Defining variables Structure of if statements, for loops, or while loops Misunderstanding the sequential reading of code	10	7
Output [O]	No output Understanding errors Not understanding errors Unexpected answer	18	15
Coding [D]	Converting known mathematics into code Expanding program to mimic mathematics	14	13

In the transcripts, <> is used to indicate an action or emotion not explicitly referenced, and [] is used to explain references and implied words used by the students. Students' references to programming code or feedback from the program are marked with **. Each excerpt is coded with a letter to indicate the category, as per Table 2 – Numerical result of analysis of transcript. A

few of the transcripts are shown with the accompanying code being discussed by the students to illustrate the origin of the discussion. However, most of the code sequences being discussed by the students are too long to be displayed properly.

Concepts were present in 18 of the segments. Within this category, recollection of the properties of a command or type occurred 11 times, making it the most prominent occurrence, with the return command of a function being the most prominent. The first excerpt reveals the student does not recall the function of a command, but the adversity is resolved with help from the group.

C-00 F: What does that *return* mean?
C-01 M: I know that.
C-02 F: What does it mean?
C-03 M: When you [the program] calculated … it [the return command] gives back the y-value [the return value], if not then it would have calculated it and … only the PC would know it [as in it is not possible to use the result later in the program], but now it has calculated and gives back [to the program] what it [the y-value] is.

Student F was looking at an example applying the use of a function and trying to understand and replicate the structure but did not fully understand the return command (C-00). Student M contributes with an understanding of the command (C-03). Conversations concerning the return command occurred in several lessons and, unlike in the above transcript, did not always lead to a resolution. Students unable to recall a specific concept slightly reframe their question. When they recall a command, but not its properties, such as with the return command, they often ask, "What does <command> mean/do?", but when they do not recall the command itself, they formulate questions in the form of "How do/can I <perform the actions of a command?>", as presented in the excerpt below.

C-10 M: How can I programme so that it [the program] asks the user to type in the expression?

The student, M, is trying to build a code where the program asks the user to input a polynomial function and is unable to recall the input command (C-10). All instances of concept adversity were of this type, and they were usually solved (14 out of 18). When concept adversity was not resolved, the students continued using the command or type without indicating that they had knowledge of its purpose (C-21).

C-20 F: What is *float*?
C-21 L: I do not remember, but it was written on the other [program from previous lesson].

Syntax was present in ten segments: defining variables occurred six times, making it the most prominent. The first adversity is the program returning an error where a name [variable] is not defined, as shown in the excerpt below.

S-00 M: *Error name y is not defined*, but y is, darn it, defined here [refers to program].

S-01 L: No, but that is not … yes, but you need to define y under here [points to another part of code], *y equals function value.*

Student M runs the program and receives an error that y is not defined and displays frustration, being sure that y is, in fact, defined (S-00). L then corrects the statement, pointing out that while M has defined y, the placement of the definition is wrong within the code (S-01). Each adversity regarding variables is similar: the students think they have defined a variable, but they are either using a wrong variable name or misplacing the definition within the code, indicating a misunderstanding of the sequential reading of the code, and displaying frustration (S-00).

Another syntax adversity is the placement of code segments, where the students are uncertain of where to put a new code segment. In the transcript below, the students, having plotted the graph of a function, are now adding a code asking for two x-values to calculate the zero point of the function using the bisectional method.

S-10 M: Where are we going to input this [segment of code]?
S-11 A: Yes, that's what I am thinking about. Is it after *y equals the function of x*?
S-12 M: No, because you have written *axhline* [command for drawing a horizontal axis].
S-13 A: Is it over the definition of the function?
S-14 M: It is all the way at the bottom [of the programming code].
S-15 A: It is at the bottom.
S-16 M: Yes, I think so, because we have written *axvline* [command for drawing a vertical axis] and so on at the bottom.
S-17 A: That is just the axis. That has nothing to do with what we input.
S-18 A: I think we type it after *y equals function value*.
S-19 H: Yeah, yeah, I agree.

The group of three students is discussing where to input a code segment (S-10, S-13, S-14, S-18) and their reasoning behind the placement (S-12, S-16, S-17). The last syntax adversity that occurred is the sequential reading of code relating to the structure and function of if statements and for and while loops. The excerpt below shows students building a program using a while loop to find a good numerical estimate for a zero point.

S-20 L: Because I have used the *while* loop, so now it keeps …
S-21 F: … keeps [running] until it finds the zero point.
S-22 L: Yes, and it looks like it is correct if you look here [refers to correlation between the output and the plot of the graph], but what is kind of stupid is that we have three zero points.
S-23 F: Oh.
S-24 L: Let's try again and see if we can find the other zero points.
S-25 F: And how do you propose we do that?
S-26 L: I think that we take like one and then <runs program and types in 1 as a starting guess>.
S-27 L: Yeah! <surprised voice> 2.16 [the correct x-value for the zero point].
S-28 L: We have managed to find the zero points.

S-29 F: But we want to get all of them at the same time, but it is not possible … because now we have guessed the three zero points. Is there no way we can guess and then receive all the zero points?

The program runs as it should (S-22, S-27), but the cubic function the group investigates has three zero points, and the program is only able to find one zero point at a time (S-29). In the final statement of the excerpt (S-29), the student evaluates the program and initiates a group discussion of ways to receive all zero points at the same time (not displayed). Syntax adversities were solved seven out of 10 times. When syntax adversity was not resolved, the students either gave up or deleted the code segment and started over.

Output adversities consist of adversities occurring due to the feedback from the editor as the students execute their program. Students understanding (six instances) or not understanding (seven instances) the error they receive represents 13 of the 18 occurrences of output adversities. When a student does not understand the error (as seen in O-00), they display frustration and openly say that they do not understand what has happened. In the excerpt below, the student continues to get an error when running the program.

O-00 L: Oh <swear word>.
O-01 L: It is just exciting to watch me fail again and again.
O-02 L: I do not understand what is wrong, but there is something wrong. It says *invalid syntax*.
O-03 M: Oh, then you have a parenthesis wrong.

Student L displays frustration and does not seem to know how to handle a syntax error (O-02). Student M then offers a possible solution to the problem (O-03), which, after a short discussion, resolves the error. On other occasions, students do understand the error, in which case they display less frustration and vocalise an idea for resolving the adversity (S-01). In the transcript below, the students are building a program to solve a quadratic equation and receive an error due to a negative value in the discriminator (O-10). One student quickly suggests the implementation of an if statement to investigate the sign of the discriminator prior to using the quadratic formula (O-11). This is evident from the following actions taken by the student, where the above if statement was coded (not shown in transcript, but in video recording of the screen).

O-10 M: Oi *x1 equals nan*, *x2 nan*.
O-11 M: So therefore, we should perhaps include an *if* [statement].

There is also the possibility of an unexpected answer, when the program outputs something that the students did not anticipate (O-10 and O-22), resulting in a discussion including evaluation and exploration. The excerpt below is taken from students solving a quadratic equation using the quadratic formula.

<Student A runs a program with values of $a = 4$, $b = 0$, and $c = -49$>.
O-20 A: Does that make any sense? That makes no sense.
O-21 H: What?
O-22 A: Why? 56? What do you mean by that? It should be 3.5 which is what it is, but this is … this is very strange.

Following this excerpt is a long discussion involving the entire group about what is wrong. They eventually resolve the unexpected answer and discover that they have forgotten to put the denominator in brackets (as in $/2 \cdot a$ instead of $/(2 \cdot a)$). The final output adversity observed is when no output is yielded after running the program. In the transcript below, the students are building a program utilising Newton's method for finding the zero point of a function, have just implemented the formula, and are now running the code for the first time.

O-30 L: Ok, *input an x-value* [to initialise Newton's method] ... and then we need to plot the tangent line.
O-31 F: Eight.
O-32 L: Ok, eight.
<types eight, runs program, no output>
O-33 L: Ok, I think this works, but now we need to plot the tangent line ... we have forgotten that.
O-34 F: But can we not write here [in the function calculating the next guess] ... like … *return*?
O-35 L: *Return* <types>.
O-36 F: *xn*?
O-37 L: *xn* <types> [the result from Newton's method].
<types eight, runs program, no output>
O-38 F: If you write like seven [as a starting guess]?
<types seven, runs program, no output>
O-39 F: No.

Students L and F run their program but receive no output. They try to alter the program (O-33 – O-37) and alter the input value (O-38), but there is still no output (O-39 – O-40). The program generates no error, and the students display uncertainty about where the error in the program resides. They end up deleting the code snippet and starting over. Output adversities were solved in 15 out of 18 instances. When the students received no output, the adversity was sometimes not resolved, and the students then deleted code and wrote a new code segment.

Coding adversities consist of adversities converting a mathematical procedure into a programming segment, which occurred 14 times. Each segment differs, since each applies to a separate mathematical idea to be converted into programming code. In the first excerpt, the students have worked through an exercise using Newton's method to find the zero point by drawing tangents of x-values close to the zero point. They have also been given the formula of Newton's method and are now trying to implement the method into programming code.

D-00 F: We need to create a tangent line for the …
D-01 L: Yes, and then we can write … *plot* … no, ok.
D-02 F: or …
D-03 L: *def tangent of x* [Python code], like that and …
D-04 F: What are we going to write here [in the program function]?
D-05 L: … and that is x.
D-06 F: *x equals*.
D-07 L: *xn* , wait a little.
D-08 F: *xn*.
D-09 L: *Equals … xn minus one*? I feel this is going to fail. *Minus f of xn*.

The students are discussing how to proceed, including using the tangent (T-00) before applying Newton's formula (T-03). At the end of the transcription segment, they foresee that their code is going to fail (T-09). The ideas displayed are well founded, but the implementation into the programming editor shows uncertainty. The second transcript is taken from when the students are building a code determining whether two variables have non-identical sign values.

D-10	H:	What is a quick way to check if two numbers have the same or different signs with the help of an *if* statement? …
D-11	H:	A quick way to check if two numbers have the same or different sign …
D-12	A:	Perhaps setting the function value equal to *a* [a variable] or something, then *if a* [programming code], the function values, if *a minus b* is positive then something.
D-13	M:	It becomes 'if the sign is the same', but …
D-14	M:	There is no command [in Python] for sign, that would have been nice.
D-15	H:	That would have been digg [Norwegian slang for nice].

Student H is stating the question to the group (T-10), and they are discussing how to make the program distinguish between the sign of two values (T-12). The task indicates the use of an if statement, but no other help is given, and they know of no command that performs this procedure. The short excerpt is the start of a longer discussion over several minutes in which several mathematical procedures are brought up and evaluated. The last coding excerpt displays the challenge of expanding a working program to generalise it. The students have built a program solving the zero points of a quadratic equation using the quadratic formula and are now asked what happens when the value of the quadratic coefficient is zero.

T-20	M:	Ok, test the program for several functions. Does it always work? It always works! What happens if you test for …
T-21	A:	It always works?
T-22	M	… for a equal to zero [the a in $ax^2 + bx + c$].
T-23	A:	What did you do...?
T-24	M:	That does not work! <Emphasis on 'that'>
T-25	M:	But then … we must, oh must we do that? Bro. Then we must change the *if* statement. Then we need two *elif*'s [else if statement in Python], is that allowed?

The students are reading the task (T-20, T-22) and then display realisation that the program does not work in the situation where a equals zero (T-24). The proposed solution is the inclusion of another set of if statements, and they question whether this is possible within the code (T-25). Coding adversity was solved 13 out of 14 times. The only time a coding was not resolved was when a student confused the mathematical definition of a function with the computational use of a function.

Discussion

The analysis shows how different types of adversity manifest when students work on MPPs and the complexity of these adversities. Concept adversities are the least surprising type and are an instrumental process that rarely contributes to mathematical learning. Concept adversity was present in all lessons and spanned a range of different commands and types, all

falling under the selection barriers defined by Ko et al. (2004). The change throughout the lessons was that the concept adversity varied as the students became more comfortable with certain programming commands and structures. Differentiation between types, typically whole number (int) and decimal numbers (float), was present in the initial lessons; in the later lessons, however, it disappeared in favour of more complex commands, such as the return command. When the students tried to recall a command that performed a certain property, they often vocalised why they needed the specific command (C-10). I would argue that the recalling of properties of a given command or type (C-00 and C-20) is less advantageous than asking how to perform a certain action (C-10). The latter results from the student's exploration and evaluation of the program and what the next step in the logical order is. Concept adversities, whether resolved or not, accounted for a small part of the lessons' overall duration and were not observed to interfere with the students' mathematical progression through the task. Designing MPPs to incrementally increase the number of commands needed (Verschaffel et al., 2020a) was observed to work well. One could possibly reduce the concept adversity further by giving the students a command cheat-sheet as a handout. While concept adversity is primarily a programming adversity, syntax adversity is more closely linked to mathematics.

Syntax adversity was the least frequent type seen and consisted of the defining of variables in the program (S-00 – S-01), placement of code sequences (S-10 – S19), and the structure of if statements and loops and their placement within the code (S-20 – S29). All are closely linked to the logical and structural build of mathematical procedures, such as the simple multiplication before addition rule. Syntax adversities, when resolved (S-00 – S-01 and S-10 – S-19), were observed to facilitate an internal construction of the programming code and the understanding of each element and the relations between them (Verschaffel et al., 2020a). When the students resolved syntax adversities, they were observed to discuss and argue about the logical procedure of the program (S-10 – S-19). This construction and understanding were more evident when the sequential structure and knowledge of the mathematical task were explicit. In the task concerning the bisectional method, where the program incrementally calculated a better and better guess for the zero point, the students voiced arguments about the structure of the programming code. In the task concerning Newton's method for finding the zero point, the students were observed to be less confident. The difference between the two was that all the steps involved in the bisectional method were known, while those in Newton's method involved new configurations, such as the formula involving both the function and its derivative. The combined complexity of both new mathematics and new programming created two adversities to overcome, possibly generating an insurmountable barrier (Kirschner et al., 2006; Reiser, 2004).

The MPPs in which syntax adversities were minimal included well-known mathematical methods together with a design facilitating the transformation of the mathematical model into a model where the elements and their relation are essential for the solution (Verschaffel et al., 2020a). This indicates that when implementing programming in the mathematics classroom, it is better to apply it as a tool for deep learning when the mathematics has already been taught, rather than using it to introduce a new topic. Syntax adversities could also be alleviated by using a wider range of structural tools, such as flow diagrams and block charts, which have been shown to be good scaffolding tools for learning Python (Cabo, 2018).

Output adversities were as prominent as concept adversities but initiated a discussion over a longer time span and were observed to contain more mathematics, often combined with programming (O-10 – O-11 and O-20 – O-22). These types of adversities were resolved and therefore desirable (D'Mello & Graesser, 2014). When output adversities were not resolved within a reasonable timeframe, the students effectively gave up and deleted the code, causing the output adversity (O-30 – O-40), making them disengage from the adidactical situation due to experiencing either an ontological or didactical obstacle (Brousseau, 1997) or an insurmountable understanding or information barrier (Ko et al., 2004). Like word problems, the programming tasks need to ensure scaffolding of the problem to facilitate the students' ability to experiment and discuss the problem (Verschaffel et al., 2020b). As the design of the MPPs focused on building tasks allowing the students to explore, the number of unresolved output adversities was low (three of 18). Output adversities are difficult to mitigate since they are hard to predict, and students have limited knowledge of how to interpret feedback from the editor. The output often becomes a problem-solving situation for the student (Polya, 1957), due to the, for them, irrational behaviour of the program (Ko et al., 2004). To circumvent this becoming an ontological obstacle, orchestrating a classroom conducive to the exploration of mathematical problem solving aids in maintaining an adidactical situation. This work reveals no direct way to prevent all output adversities, as the number of possible errors is numerous; however, teaching the students basic knowledge of how to handle errors was particularly advantageous.

Coding adversity is, for the mathematically inclined, perhaps the most interesting, where the combination and advantage of mathematics and programming becomes visible. The typical sequence in coding adversity consists of the students exploring a mathematical problem or method, followed by how to solve the problem or implement the method into the program. The exploratory talk consisted of engagement (T-20 – T-21), suggestions for joint consideration (T-12, T-25), and justification (T-10 – T-15 and onwards), following the definition given by Mercer (2005). This element of exploratory talk is beneficial and contributes to learning (Mercer, 2005). The MPPs were built to take advantage of programming in mathematics using coding to solve a problem. Prior to their implementation in the mathematics classroom, tasks containing coding from mathematics to programming were viewed to be the most promising and were continuously added during the iterative design process. As seen in excerpt two from the coding adversities (T-10 – T-15), the seemingly simple problem of making the program differentiate whether two numbers have the same or a different sign became a mathematical discussion lasting several minutes and resulting in a short and simple code. The two main features of coding were to (1) use mathematics to make the program understand a simple mathematical relationship (as above), or (2) use mathematics to solve a complex problem through building a short program (bisectional method, Newton's method). Avoiding coding becoming an ontogenic obstacle, depending on the students' building of a mathematical model into a programming model where the elements and their relations are essential for the solution (Ko et al., 2004). Facilitating this building, the task needs to scaffold the intended problem using and recalling the previous mathematical knowledge required and guide the students in their exploration of how to assemble a new mathematical and/or programming model (Kirschner et al., 2006; Reiser, 2004).

Concluding thoughts

Adversities when applying programming in the mathematics classroom are inevitable, as even coding experts encounter errors from time to time. The change from requiring a single input into a digital tool such as GeoGebra to building a program consisting of several lines of code is significant. The challenge for educators to reduce the adversities, in terms of both designing MPPs and orchestrating lessons, is substantial and needs further research. The work presented indicates several ways of reducing the number of adversities the students encounter and reflects on which should remain as part of mathematical learning. By scaffolding the design (Kirschner et al., 2006; Reiser, 2004) and orchestrating a classroom encouraging mathematical exploration (Mercer, 2005), these adversities can become valuable to learning. As regards designing MPPs, I would suggest care is taken to avoid building what could possibly become insurmountable barriers. Adversities should be predictable (Stein et al., 2008), as they are linked to the task design and therefore, to some extent, preventable. Further research into the affordances and constraints of implementing programming in the classroom is needed, especially studies spanning more than one class. As Niss (1999) stated, if we know the adversities that block the paths of students learning mathematics using programming, we will gain a better understanding of how mathematical knowledge and ability can be combined with programming, leading to better learning. Programming allows for mathematical investigation into many previously unavailable themes, such as numerical methods, as well as facilitating an in-depth learning of existing school mathematics, such as function analysis. To avoid the same fate as that seen the last time programming was implemented (Misfeldt & Ejsing-Duun, 2015; Papert, 1980), research into how it can facilitate mathematical learning is needed.

References

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, *3*(2), 115–138.

Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction*, *16*, 68–76.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). The Nordic approach to introducing Computational Thinking and programming in compulsory education. *Report prepared for the Nordic@ BETT2018 Steering Group*, 397–400.

Bosse, Y., & Gerosa, M. A. (2017). Why is programming so difficult to learn? Patterns of difficulties related to programming learning mid-stage. *ACM SIGSOFT Software Engineering Notes*, *41*(6), 1–6.

Brousseau, G. (1997). *Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990* (B. N. Cooper M, Sutherland R, Warfield V, Trans.). Kluwer Academic Publishers.

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, *87*(4), 834–860.

Cabo, C. (2018). Effectiveness of flowcharting as a scaffolding tool to learn Python. 2018 IEEE Frontiers in Education Conference (FIE).

Campbell, J. I., & Epp, L. J. (2005). Architectures for arithmetic. In J.I.D. Campbell (Ed.) *Handbook of Mathematical Cognition*, 347-360. Psychology Press.

D'Mello, S., Lehman, B., Pekrun, R., & Graesser, A. (2014). Confusion can be beneficial for learning. *Learning and Instruction*, *29*, 153–170.

D'Mello, S. K., & Graesser, A. C. (2014). Confusion. In R. Pekrun, & L. Linnenbrink-Garcia (Eds.), *International Handbook of Emotions in Education* (pp. 299–320). Routledge.

Daroczy, G., Wolska, M., Meurers, W. D., & Nuerk, H.-C. (2015). Word problems: a review of linguistic and numerical factors contributing to their difficulty. *Frontiers in Psychology*, *6*, 348.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

Jenkins, T. (2002). On the difficulty of learning to program. Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences,

Kirschner, P., Sweller, J., & Clark, R. E. (2006). Why unguided learning does not work: An analysis of the failure of discovery learning, problem-based learning, experiential learning and inquiry-based learning. *Educational Psychologist*, *41*(2), 75–86.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. 2004 IEEE Symposium on Visual Languages-Human Centric Computing,

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *Acm Sigcse Bulletin*, *37*(3), 14–18.

Lodge, J. M., Kennedy, G., Lockyer, L., Arguel, A., & Pachman, M. (2018). Understanding difficulties and resulting confusion in learning: an integrative review. *Frontiers in Education, 3, 49*

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, *62*(2), 77–90.

Mercer, N. (2005). Sociocultural discourse analysis: Analysing classroom talk as a social mode of thinking. *Journal of Applied Linguistics and Professional Practice*, *1*(2), 137–168.

Merenluoto, K., & Lehtinen, E. (2004). Number concept and conceptual change: Towards a systemic model of the processes of change. *Learning and Instruction*, *14*(5), 519–534.

Misfeldt, M., & Ejsing-Duun, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. In K. Krainer & N. Vondrová (Eds.), *CERME9: Proceedings of the Ninth Congress of the European Society for Research in Mathematics Education (pp. 2524–2530)*. Prague, Czech Republic: Charles University in Prague, Faculty of Education and ERME

Muir, T., Beswick, K., & Williamson, J. (2008). "I'm not very good at solving problems": An exploration of students' problem solving behaviours. *The Journal of Mathematical Behavior*, *27*(3), 228–241.

Nelson, G., & Powell, S. R. (2018). A systematic review of longitudinal studies of mathematics difficulty. *Journal of Learning Disabilities*, *51*(6), 523–539.

Niss, M. (1999). Aspects of the nature and state of research in mathematics education. *Educational Studies in Mathematics*, *40*(1), 1–24.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Piteira, M., & Costa, C. (2013). Learning computer programming: study of difficulties in learning programming. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication (pp. 75–80)*. New York, NY, USA: ACM

Polya, G. (1957). *How to solve it: A new aspect of mathematical methods*. Prentice University Press.

Pongsakdi, N., Kajamies, A., Veermans, K., Lertola, K., Vauras, M., & Lehtinen, E. (2020). What makes mathematical word problem solving challenging? Exploring the roles of word problem characteristics, text comprehension, and arithmetic skills. *Zdm*, *52*(1), 33–44.

Reiser, B. J. (2004). Scaffolding complex learning: The mechanisms of structuring and problematizing student work. *The Journal of the Learning Sciences*, *13*(3), 273–304.

Ryve, A. (2011). Discourse research in mathematics education: A critical evaluation of 108 journal articles. *Journal for Research in Mathematics Education*, *42*(2), 167–199.

Stein, M. K., Engle, R. A., Smith, M. S., & Hughes, E. K. (2008). Orchestrating productive mathematical discussions: Five practices for helping teachers move beyond show and tell. *Mathematical Thinking and Learning*, *10*(4), 313–340.

Stephens, M., & Kadijevich, D. M. (2020). Computational/algorithmic thinking. In Lerman, S. (Ed.), *Encyclopedia of Mathematics Education*, 117–123. Dordrecht, the Netherlands

Verschaffel, L., Depaepe, F., & Van Dooren, W. (2020a). Word problems in mathematics education. In Lerman, S. (Ed.), *Encyclopedia of Mathematics Education*, 908-911. Dordrecht, the Netherlands

Verschaffel, L., Greer, B., & De Corte, E. (2000). *Making sense of word problems*. Lisse: Swets&Zeitlinger.

Verschaffel, L., Schukajlow, S., Star, J., & Van Dooren, W. (2020b). Word problems in mathematics education: A survey. *Zdm*, *52*(1), 1–16.

Wegerif, R. (2007). *Dialogic education and technology: Expanding the space of learning* (Vol. 7). Springer Science & Business Media.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *Acm Sigcse Bulletin*, *28*(3), 17–22.

ISBN: 978-82-575-2004-5
ISSN: 1894-6402

Norwegian University
of Life Sciences

Postboks 5003
NO-1432 Ås, Norway
+47 67 23 00 00
www.nmbu.no

115012 / ANDVORDGRAFISK.NO